IDL Quick Reference

RSI

IDL Version 6.2
July 2005 Edition

0705IDL62QR

Restricted Rights Notice

The IDL®, ION Script™, and ION Java™ software programs and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. RSI reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

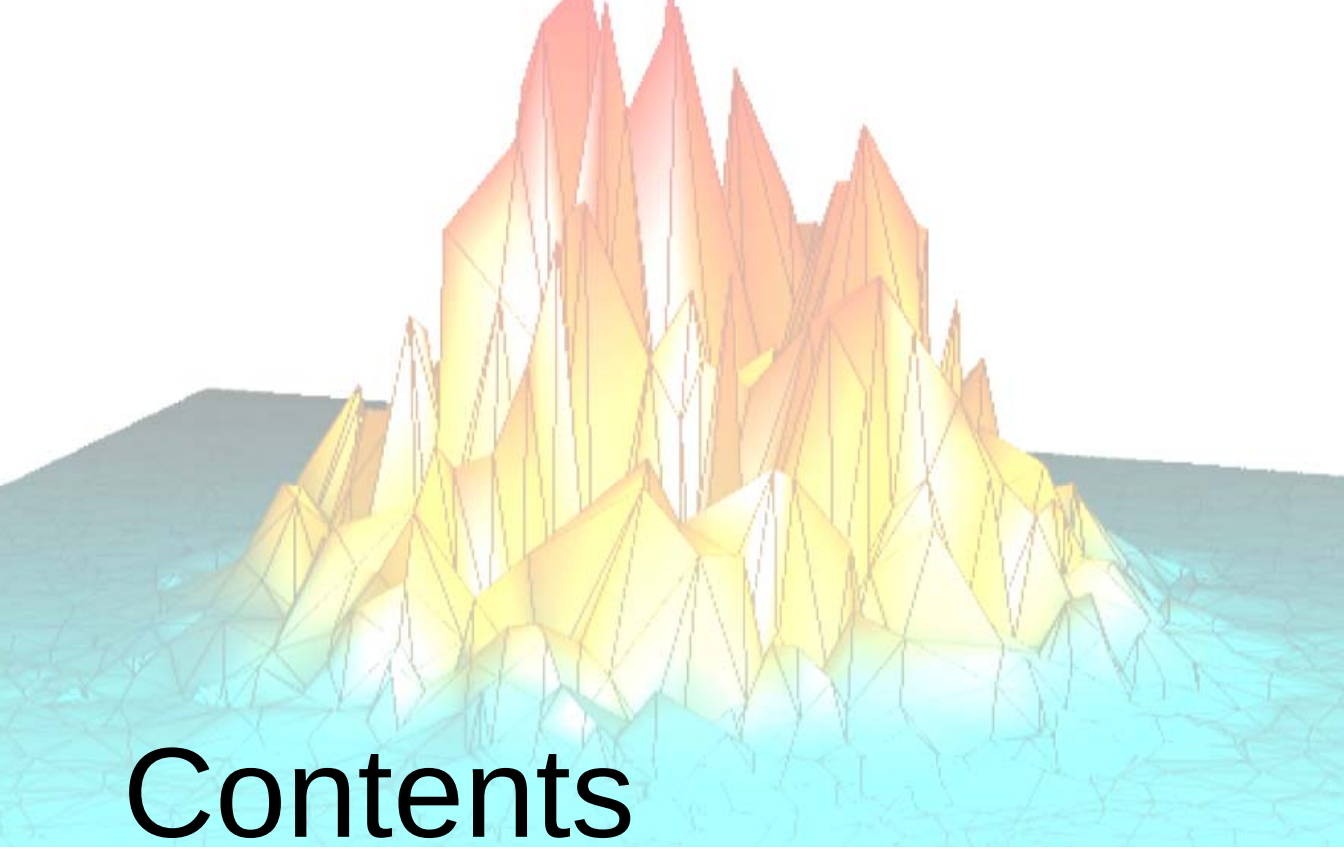
RSI makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

RSI shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the IDL or ION software packages or their documentation.

Permission to Reproduce this Manual

If you are a licensed user of this product, RSI grants you a limited, nontransferable license to reproduce this particular document provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments



Contents

Functional List of IDL Routines

The following is a list of all routines included in IDL, categorized by functionality.

3D Visualization

3D Transformations & Scene Setup

CONVERT_COORD - Transforms coordinates to and from the coordinate systems supported by IDL.

COORD2TO3 - Returns 3D data coordinates given normalized screen coordinates.

CREATE_VIEW - Sets up 3D transformations.

CV_COORD - Converts 2D and 3D coordinates between coordinate systems.

IDLgrLight - Represents a source of illumination for three-dimensional graphic objects.

IDLgrModel - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated).

IDLgrScene - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects.

IDLgrView - Represents a rectangular area in which graphics objects are drawn.

IDLgrViewgroup - Represents a simple container object, very similar to the IDLgrScene object, but can contain objects without a Draw method in addition to IDLgrView objects.

IDLgrWindow - Represents an on-screen area on a display device that serves as a graphics destination.

SCALE3 - Sets up axis ranges and viewing angles for 3D plots.

SCALE3D - Scales 3D unit cube into the viewing area.

SET_SHADING - Sets the light source shading parameters.

SURFR - Sets up 3D transformations by duplicating rotation, translation, and scaling of SURFACE.

T3D - Performs various 3D transformations.

VERT_T3D - Transforms a 3D array by a 4x4 transformation matrix.

VOXEL_PROJ - Generates volume visualizations using voxel technique.

Polygonal Mesh Routines

COMPUTE_MESH_NORMALS - Computes normal vectors for a set of polygons described by the input array.

IDLgrPolygon - Represents one or more polygons that share a given set of vertices and rendering attributes.

MESH_CLIP - Clips a polygonal mesh to an arbitrary plane in space and returns a polygonal mesh of the remaining portion.

MESH_DECIMATE - Reduces the density of geometry while preserving as much of the original data as possible.

MESH_ISSOLID - Computes various mesh properties and enables IDL to determine if a mesh encloses space (is a solid).

MESH_MERGE - Merges two polygonal meshes.

MESH_NUMTRIANGLES - Computes the number of triangles in a polygonal mesh.

MESH_OBJ - Generates a polygon mesh for various simple objects.

MESH_SMOOTH - Performs spatial smoothing on a polygon mesh.

MESH_SURFACEAREA - Computes various mesh properties to determine the mesh surface area, including integration of other properties interpolated on the surface of the mesh.

MESH_VALIDATE - Checks for NaN values in vertices, removes unused vertices, and combines close vertices.

MESH_VOLUME - Computes the volume that the mesh encloses.

POLYSHADE - Creates a shaded surface representation from a set of polygons.

Surfaces and Contours

CONTOUR - Draws a contour plot.

ICONTOUR - Creates an iTool and associated user interface (UI) configured to display and manipulate contour data.

IDLgrContour - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points.

IDLgrSurface - Represents a shaded or vector representation of a mesh grid.

IMAGE_CONT - Overlays an image with a contour plot.

ISURFACE - Creates an iTool and associated user interface (UI) configured to display and manipulate surface data.

MIN_CURVE_SURF - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.

POLAR_CONTOUR - Draws a contour plot from data in polar coordinates.

SHADE_SURF - Creates a shaded-surface representation of gridded data.

SHADE_SURF_IRR - Creates a shaded-surface representation of an irregularly gridded dataset.

SHOW3 - Displays array as image, surface plot, and contour plot simultaneously.

SURFACE - Plots an array as a wireframe mesh surface.

XSURFACE - Provides GUI to SURFACE and SHADE_SURF.

Tetrahedral Mesh Routines

IDLgrTessellator - Decomposes a polygon description into a set of triangles to convert complex and/or concave polygons into a convex form suitable for drawing with the IDLgrPolygon object.

TETRA_CLIP - Clips a tetrahedral mesh to an arbitrary plane in space and returns a tetrahedral mesh of the remaining portion.

TETRA_SURFACE - Extracts a polygonal mesh as the exterior surface of a tetrahedral mesh.

TETRA_VOLUME - Computes properties of tetrahedral mesh array.

Vector Field Visualization

FLOW3 - Draws lines representing a 3D flow/velocity field.

INTERPOL - Performs linear interpolation on vectors.

PARTICLE_TRACE - Traces the path of a massless particle through a vector field.

STREAMLINE - Generates the visualization graphics from a path.

VECTOR_FIELD - Places colored, oriented vectors of specified length at each vertex in an input vertex array.

VEL - Draws a velocity (flow) field with streamlines.

VELOVECT - Draws a 2D velocity field plot.

Volume Visualization

EXTRACT_SLICE - Returns 2D planar slice extracted from volume.

IDLgrVolume - Represents a mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions.

IDLgrVRML - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.

INTERVAL_VOLUME - Generates a tetrahedral mesh from volumetric data.

ISOSURFACE - Returns topologically consistent triangles by using oriented tetrahedral decomposition.

IVOLUME - Creates an iTool and associated user interface (UI) configured to display and manipulate volume data.

PROJECT_VOL - Returns a translucent rendering of a volume projected onto a plane.

QGRID3 - Interpolates the dependent variable values to points in a regularly sampled volume.

QHULL - Constructs convex hulls, Delaunay triangulations, and Voronoi diagrams.

RECON3 - Reconstructs a 3D representation of an object from 2D images.

SEARCH3D - Finds “objects” or regions of similar data values within a volume.

SHADE_VOLUME - Contours a volume to create a list of vertices and polygons that can be displayed using POLYSHADE.

SLICER3 - Interactive volume visualization tool.

VOXEL_PROJ - Generates volume visualizations using voxel technique.

XOBJVIEW - Displays object viewer widget.

XOBJVIEW_ROTATE - Programmatically rotate the object currently displayed in XOBJVIEW.

XOBJVIEW_WRITE_IMAGE - Write the object currently displayed in XOBJVIEW to an image file.

XVOLUME - Utility for viewing and interactively manipulating volumes and isosurfaces.

Animation

CW_ANIMATE - Creates a compound widget for animation.

CW_ANIMATE_GETP - Gets pixmap window IDs used by CW_ANIMATE.

CW_ANIMATE_LOAD - Loads images into CW_ANIMATE.

CW_ANIMATE_RUN - Displays images loaded into CW_ANIMATE.

IDLgrModel - Animates the display of objects by displaying a single object in the collection of objects contained in the model when the RENDER_METHOD property is set.

IDLgrMPEG - Creates an MPEG movie file from an array of image frames.

FLICK - Causes the display to flicker between two images.

MPEG_CLOSE - Closes an MPEG sequence.

MPEG_OPEN - Opens an MPEG sequence.

MPEG_PUT - Inserts an image array into an MPEG sequence.

MPEG_SAVE - Saves an MPEG sequence to a file.

XINTERANIMATE - Displays animated sequence of images.

Array Creation

BINDGEN - Returns byte array with each element set to its subscript.

BYTARR - Creates a byte vector or array.

CINDGEN - Returns a complex array with each element set to its subscript.

COMPLEXARR - Creates a complex, single-precision, floating-point vector or array.

DBLARR - Creates a double-precision array.

DCINDGEN - Returns a double-precision, complex array with each element set to its subscript.

DCOMPLEXARR - Creates a complex, double-precision vector or array.

DINDGEN - Returns a double-precision array with each element set to its subscript.

FINDGEN - Returns a floating-point array with each element set to its subscript.

FLTARR - Returns a single-precision, floating-point vector or array.

IDENTITY - Returns an identity array (an array with ones along the main diagonal and zeros elsewhere) of the specified dimensions.

INDGEN - Return an integer array with each element set to its subscript.

INTARR - Creates an integer vector or array.

L64INDGEN - Returns a 64-bit integer array with each element set to its subscript.

LINDGEN - Returns a longword integer array with each element set to its subscript.

LON64ARR - Returns a 64-bit integer vector or array.

LONARR - Returns a longword integer vector or array.

MAKE_ARRAY - Returns an array of the specified type, dimensions, and initialization.

OBJARR - Creates an array of object references.

PTRARR - Creates an array of pointers.

REPLICATE - Creates an array of given dimensions, filled with specified value.

SINDGEN - Returns a string array with each element set to its subscript.

STRARR - Returns string array containing zero-length strings.

TIMEGEN - Returns an array of double-precision floating-point values that represent times in Julian dates.

UINDGEN - Returns unsigned integer array with each element set to its subscript.

UINTARR - Returns an unsigned integer vector or array.

UL64INDGEN - Returns an unsigned 64-bit integer array with each element set to its subscript.

ULINDGEN - Returns an unsigned longword array with each element set to its subscript.

ULON64ARR - Returns an unsigned 64-bit integer vector or array.

ULONARR - Returns an unsigned longword integer vector or array.

Array Manipulation

ARRAY_EQUAL - Provides fast test for data equality in cases where the positions of the differing data elements is not required.

ARRAY_INDICES - Converts one-dimensional subscripts of an array into corresponding multi-dimensional subscripts.

BLAS_AXPY - Updates existing array by adding a multiple of another array.

INVERT - Computes the inverse of a square array.

MAX - Returns the value of the largest element of Array.

MEDIAN - Returns the median value of Array or applies a median filter.

MIN - Returns the value of the smallest element of an array.

REFORM - Changes array dimensions without changing the total number of elements.

REPLICATE_INPLACE - Updates an array by replacing all or selected parts of it with a specified value.

REVERSE - Reverses the order of one dimension of an array.

ROT - Rotates an image by any amount.

ROTATE - Rotates/transposes an array in multiples of 90 degrees.

SHIFT - Shifts elements of vectors or arrays by a specified number of elements.

SIZE - Returns array size and type information.

SORT - Returns indices of an array sorted in ascending order.

TOTAL - Sums of the elements of an array.

TRANSPOSE - Transposes an array.

UNIQ - Returns subscripts of the unique elements in an array.

WHERE - Returns subscripts of nonzero array elements.

XVAREDIT - Provides widget-based editor for IDL variables.

Color Table Manipulation

CMYK_CONVERT - Converts color triples to and from RGB and CMYK.

COLOR_CONVERT - Converts color triples to and from RGB, HLS, and HSV.

COLOR_QUAN - Converts true-color (24-bit) image to pseudo-color (8-bit) image.

COLORMAP_APPLICABLE - Determines whether the current visual class supports the use of a colormap.

CT_LUMINANCE - Calculates the luminance of colors.

CW_PALETTE_EDITOR - Creates compound widget to display and edit color palettes.

CW_PALETTE_EDITOR_GET - Gets CW_PALETTE_EDITOR properties.

CW_PALETTE_EDITOR_SET - Sets CW_PALETTE_EDITOR properties.

GAMMA_CT - Applies gamma correction to a color table.

H_EQ_CT - Histogram-equalizes the color tables for an image or a region of the display.

H_EQ_INT - Interactively histogram-equalizes the color tables of an image or a region of the display.

HLS - Creates color table in Hue, Lightness, Saturation color system.

HSV - Creates color table based on Hue and Saturation Value color system.

IDLgrPalette - Represents a color lookup table that maps indices to red, green, and blue values.

LOADCT - Loads one of the predefined IDL color tables.

MODIFYCT - Saves modified color tables in the IDL color table file.

MULTI - Replicates current color table to enhance contrast.

PSEUDO - Creates pseudo-color table based on Lightness, Hue, and Brightness system.

REDUCE_COLORS - Reduces the number of colors used in an image by eliminating unused pixel values.

STRETCH - Stretches color table for contrast enhancement.

TEK_COLOR - Loads color table based on Tektronix printer.

TVLCT - Loads display color tables.

XLOADCT - Provides GUI to interactively select and load color tables.

XPALETTE - Displays widget used to create and modify color tables.

Date and Time

BIN_DATE - Converts ASCII date/time string to binary string.

CALDAT - Converts Julian date to month, day, year.

CALENDAR - Displays a calendar for a given month or year.

JULDAY - Returns Julian Day Number for given month, day, and year.

SYSTIME - Returns the current time as either a date/time string, as the number of seconds elapsed since 1 January 1970, or as a Julian date/time value.

TIMEGEN - Returns an array of double-precision floating-point values that represent date/times in terms of Julian values.

Debugging

.CONTINUE - Continues execution of a stopped program.

.SKIP - Skips over the next n statements and then single steps.

.STEP - Executes one or n statements from the current position.

.STEPOVER - Executes a single statement if the statement doesn't call a routine.

.TRACE - Similar to .CONTINUE, but displays each line of code before execution.

BREAKPOINT - Sets and clears breakpoints for debugging.

SHMDEBUG - Print debugging information when a variable loses reference to an underlying shared memory segment.

STOP - Stops the execution of a running program or batch file.

Dialog Routines

DIALOG_MESSAGE - Creates modal message dialog.

DIALOG_PICKFILE - Creates native file-selection dialog.

DIALOG_PRINTERSETUP - Opens native dialog used to set properties for a printer.

DIALOG_PRINTJOB - Opens native dialog used to set parameters for a print job.

DIALOG_READ_IMAGE - Presents GUI for reading image files.

DIALOG_WRITE_IMAGE - Presents GUI for writing image files.

Direct Graphics

ANNOTATE - Starts IDL widget used to interactively annotate images and plots with text and drawings.

ARROW - Draws line with an arrow head.

AXIS - Draws an axis of the specified type and scale at a given position.

BAR_PLOT - Creates a bar graph.

BOX_CURSOR - Emulates operation of a variable-sized box cursor.

CONVERT_COORD - Transforms coordinates to and from the coordinate systems supported by IDL.

CONTOUR - Draws a contour plot.

CREATE_VIEW - Sets the various system variables required to define a coordinate system and a 3-D view.

CURSOR - Reads position of the interactive graphics cursor.

CVTTOBM - Creates a bitmap byte array for a button label.

DEFROI - Defines an irregular region of interest of an image using the image display system and the cursor and mouse.

DEVICE - Sets to plot in device coordinates.

DRAW_ROI - Draws a region or group of regions to the current Direct Graphics device.

EMPTY - Empties the graphics output buffer.

ERASE - Erases the screen of the current graphics device, or starts a new page if the device is a printer.

ERRPLOT - Plots error bars over a previously drawn plot.

FLICK - Causes the display to flicker between two output images at a given rate.

FORMAT_AXIS_VALUES - Formats numbers as strings for use as axis values.

IMAGE_CONT - Overlays an image with a contour plot.

LABEL_REGION - Consecutively labels all of the regions, or blobs, of a bi-level image with a unique region index.

LOADCT - Loads a pre-defined color table.

OPLOT - Plots vector data over a previously drawn plot.

OPLOTERR - Plots error bars over a previously drawn plot.

PLOT - Draws a graph of vector arguments.

PLOT_3DBOX - Plots a function of two variables (e.g., Z=*f*(X, Y)) inside a 3-D box.

PLOT_FIELD - Plots a 2-D field. N random points are picked, and from each point a path is traced along the field.

PLOTERR - plots individual data points with error bars.

PLOTS - Plots vectors and points.

POLAR_CONTOUR - Draws a contour plot from data in polar coordinates.

POLAR_SURFACE - Interpolates a surface from polar coordinates (R, Theta, Z) to rectangular coordinates (X, Y, Z).

POLYFILL - Fills the interior of a polygon.

POLYSHADE - Creates a shaded-surface representation of one or more solids described by a set of polygons.

PROFILE - Extracts a profile from an image.

PROFILES - Interactively examines image profiles.

PROJECT_VOL - Returns a two-dimensional image that is the projection of a 3-D volume of data onto a plane (similar to an X-ray).

RDPIX - Interactively displays the X position, Y position, and pixel value at the cursor.

SCALE3 - Sets up transformation and scaling parameters for basic 3-D viewing.

SCALE3D - Scales the 3-D unit cube (a cube with the length of each side equal to 1) into the viewing area.

SET_PLOT - Sets the output device used by the IDL direct graphics procedures.

SET_SHADING - Modifies the light source shading parameters that affect the output of SHADE_SURF and POLYSHADE.

SHADE_SURF - Creates a shaded-surface representation of a regular or nearly-regular gridded surface.

SHADE_SURF_IRR - Creates a shaded-surface representation of an irregularly gridded elevation dataset. Given a 3-D volume and a contour value, SHADE_VOLUME produces a list of vertices and polygons describing the contour surface.

SHADE_VOLUME - Given a 3-D volume and a contour value, produces a list of vertices and polygons describing the contour surface.

SHOW3 - Combines an image, a surface plot of the image data, and a contour plot of the images data in a single tri-level display.

SURFACE - draws a wire-mesh representation of a two-dimensional array projected into two dimensions, with hidden lines removed.

THREED - Plots a 2D array as a pseudo 3D plot.

TV - Displays an image.

TVCRS - Manipulates the image display cursor.

TVLCT - Loads a predefined color table or a color table from specified variables.

TVRD - Returns the contents of the specified rectangular portion of the current graphics window or device.

TVSCL - Scales and displays an image.

WINDOW - Creates a window for the display of graphics or text.

WSET - Selects the current window.

WSHOW - Exposes or hides the designated window.

XYOUTS - Draws text on currently-selected graphics device.

ZOOM - Zooms portions of the display.

ZOOM_24 - Zooms portions of true-color (24-bit) display.

Error Handling

CATCH - Intercepts and processes error messages, and continues program execution.

MESSAGE - Issues error and informational messages.

ON_ERROR - Designates the error recovery method.

ON_IOERROR - Declares I/O error exception handler.

STRMESSAGE - Returns the text of a given error number.

Executive Commands

.COMPILE - Compiles programs without running.

.CONTINUE - Continues execution of a stopped program.

.EDIT - Opens files in editor windows of the IDLDE (Windows and Motif only).

.FULL_RESET_SESSION - Does everything .RESET_SESSION does, plus additional reset tasks such as unloading sharable libraries.

.GO - Executes a previously compiled $MAIN$ program.

.OUT - Continues execution until the current routine returns.

.RESET_SESSION - Resets much of the state of an IDL session without requiring the user to exit and restart the IDL session.

.RETURN - Continues execution until RETURN statement.

.RNEW - Erases $MAIN$ program variables and then executes .RUN.

.RUN - Compiles and executes IDL commands from files or keyboard.

.SKIP - Skips over the next n statements and then single steps.

.STEP - Executes one or n statements from the current position.

.STEPOVER - Executes a single statement if the statement does not call a routine.

.TRACE - Similar to .CONTINUE, but displays each line of code before execution.

External Linking

CALL_EXTERNAL - Calls a function in an external sharable object and returns a scalar value.

DLM_LOAD - Explicitly causes a DLM to be loaded.

IDLcomActiveX - Creates an IDL object that encapsulates an ActiveX control.

IDLcomIDispatch - Creates an IDL object that encapsulates a COM object.

IDLjavaObject - An IDL object encapsulating a Java object. IDL provides data type and other translation services, allowing IDL programs to access the Java object's methods and properties using standard IDL syntax.

LINKIMAGE - Merges routines written in other languages with IDL at run-time.

MAKE_DLL - Compiles and links sharable libraries (DLLs).

Font Manipulation

EFONT - Interactive vector font editor and display tool.

IDLgrFont - represents a typeface, style, weight, and point size that may be associated with text objects.

PS_SHOW_FONTS - Displays all the PostScript fonts that IDL knows about.

PSAFM - Converts Adobe Font Metrics file to IDL format.

SHOWFONT - Displays a TrueType or vector font

XFONT - Creates modal widget to select and view an X Windows font.

Help Routines

? - Invokes the IDL Online Help facility when entered at the IDL command line.

DOC_LIBRARY - Extracts documentation headers from IDL programs.

HELP - Provides information about the current IDL session.

MEMORY - Returns information about dynamic memory currently in use by the IDL session.

MK_HTML_HELP - Converts text documentation headers to HTML files.

ONLINE_HELP - Invokes online help viewer from programs.

STRUCT_HIDE - Prevents the IDL HELP procedure from displaying information about structures or objects.

Image Processing

Contrast Enhancement and Filtering

ADAPT_HIST_EQUAL - Performs adaptive histogram equalization

BYTSCL - Scales all values of an array into range of bytes.

CONVOL - Convolves two vectors or arrays.

DIGITAL_FILTER - Calculates coefficients of a non-recursive, digital filter.

FFT - Returns the Fast Fourier Transform of an array.

HILBERT - Constructs a Hilbert transform.

HIST_EQUAL - Histogram-equalizes an image.

LEEFILT - Performs the Lee filter algorithm on an image array.

MEDIAN - Returns the median value of Array or applies a median filter.

ROBERTS - Returns an approximation of Roberts edge enhancement.

SMOOTH - Smooths with a boxcar average.

SOBEL - Returns an approximation of Sobel edge enhancement.

UNSHARP_MASK - Performs an unsharp-mask sharpening filter on a two-dimensional array or a truecolor image.

See Also - Wavelet Toolkit

Feature Extraction/Image Segmentation

CONTOUR - Draws a contour plot.

DEFROI - Defines an irregular region of interest of an image.

HISTOGRAM - Computes the density function of an array.

HOUGH - Returns the Hough transform of a two-dimensional image.

IMAGE_STATISTICS - Computes sample statistics for a given array of values.

ISOCONTOUR - Interprets the contouring algorithm found in the IDLgrContour object.

ISOSURFACE - Returns topologically consistent triangles by using oriented tetrahedral decomposition.

LABEL_REGION - Labels regions (blobs) of a bi-level image.

MAX - Returns the value of the largest element of Array.

MEDIAN - Returns the median value of Array or applies a median filter.

MIN - Returns the value of the smallest element of an array.

PROFILES - Interactively examines image profiles.

RADON - Returns the Radon transform of a two-dimensional image.

REGION_GROW - Perform region growing.

SEARCH2D - Finds "objects" or regions of similar data within a 2D array.

THIN - Returns the "skeleton" of a bi-level image.

UNIQ - Returns subscripts of the unique elements in an array.

WATERSHED - Applies the morphological watershed operator to a grayscale image.

WHERE - Returns subscripts of nonzero array elements.

Image Display

DISSOLVE - Provides a digital "dissolve" effect for images.

IDLgrImage - Creates an image object that represents a mapping from a 2D array of data values to a 2D array of pixel colors.

IDLgrPalette - Represents a color lookup table that maps indices to red, green, and blue values.

IIMAGE - Creates an iTool and associated user interface (UI) configured to display and manipulate image data.

RDPIX - Interactively displays image pixel values.

SLIDE_IMAGE - Creates a scrolling graphics window for examining large images.

TV - Displays an image. To scale and display the image, use TVSCL.

TVCRS - Manipulates the image display cursor.

TVLCT - Loads display color tables.

TVSCL - Scales and displays an image.

XOBJVIEW - Displays object viewer widget.

XOBJVIEW_ROTATE - Programmatically rotate the object currently displayed in XOBJVIEW.

XOBJVIEW_WRITE_IMAGE - Write the object currently displayed in XOBJVIEW to an image file.

ZOOM - Zooms portions of the display.

ZOOM_24 - Zooms portions of true-color (24-bit) display.

Image Geometry Transformations

CONGRID - Resamples an image to any dimensions.

EXPAND - Shrinks/expands image using bilinear interpolation.

EXTRAC - Returns sub-matrix of input array. Array operators (e.g., * and :) should usually be used instead.

INTERPOLATE - Returns an array of interpolates.

INVERT - Computes the inverse of a square array.

POLY_2D - Performs polynomial warping of images.

POLYWARP - Performs polynomial spatial warping.

REBIN - Resizes a vector or array by integer multiples.

REFORM - Changes array dimensions without changing the total number of elements.

REVERSE - Reverses the order of one dimension of an array.

ROT - Rotates an image by any amount.

ROTATE - Rotates/transposes an array in multiples of 90 degrees.

SHIFT - Shifts elements of vectors or arrays by a specified number of elements.

TRANSPOSE - Transposes an array.

WARP_TRI - Warps an image using control points.

Morphological Image Operators

DILATE - Implements morphologic dilation operator on binary and grayscale images.

ERODE - Implements the erosion operator on binary and grayscale images and vectors.

LABEL_REGION - Labels regions (blobs) of a bi-level image.

MORPH_CLOSE - Applies closing operator to binary or grayscale image.

MORPH_DISTANCE - Estimates N-dimensional distance maps, which contain for each foreground pixel the distance to the nearest background pixel, using a given norm.

MORPH_GRADIENT - Applies the morphological gradient operator to a grayscale image.

MORPH_HITORMISS - Applies the hit-or-miss operator to a binary image.

MORPH_OPEN - Applies the opening operator to a binary or grayscale image.

MORPH_THIN - Performs a thinning operation on binary images.

MORPH_TOPHAT - Applies top-hat operator to a grayscale image.

WATERSHED - Applies the morphological watershed operator to a grayscale image.

Regions of Interest

CW_DEFROI - Creates compound widget used to define region of interest.

DEFROI - Defines an irregular region of interest of an image.

DRAW_ROI - Draws region or group of regions to current Direct Graphics device.

IDLanROI - Represents a region of interest used for analysis.

IDLanROIGroup - Analytical representation of a group of regions of interest.

IDLgrROI - Object graphics representation of a region of interest.

IDLgrROIGroup - Object Graphics representation of a group of regions of interest.

LABEL_REGION - Labels regions (blobs) of a bi-level image.

REGION_GROW - Grows an initial region to include all areas that match specified constraints.

XROI - Utility for defining regions of interest, and obtaining geometry and statistical data about these ROIs.

Input/Output

General File Access

APP_USER_DIR - Provides access to the *application user directory.*

APP_USER_DIR_QUERY - Locates existing *application user directories.*

DIALOG_PICKFILE - Creates native file-selection dialog.

FILE_BASENAME - Returns the final segment of a file path.

FILE_CHMOD - Changes the current access permission associated with a file or directory.

FILE_COPY - Copies files or directories to a new location.

FILE_DELETE - Removes a file or empty directory.

FILE_DIRNAME - Returns the directory name of a file path consisting of everything except the final segment of the file path.

FILE_EXPAND_PATH - Returns a fully qualified path regardless of the current working directory.

FILE_INFO - Returns status information about a file.

FILE_LINES - Returns the number of lines of text in a file.

FILE_LINK - Creates Unix file links.

FILE_MKDIR - Creates a new directory or directories.

FILE_MOVE - Renames files and directories.

FILE_POLL_INPUT - Blocks processing until it detects that a read operation on a specified file will succeed.

FILE_READLINK - Returns the path pointed to by a Unix symbolic link.

FILE_SAME - Determines it two different file names refer to the same underlying file.

FILE_SEARCH - Returns a string array containing the names of all files matching the input path specification.

FILE_TEST - Test a file or directory for existence and other specific attributes.

FILE_WHICH - Returns the path for the first file for the given name found by searching the specified path.

FILEPATH - Returns full path to a file in the IDL distribution.

FSTAT - Returns information about a specified file unit.

PATH_SEP - Returns the proper file path segment separator character for the current operating system.

Image Data Formats

NOTE: Also see "Query Routines" on page 21.

DIALOG_READ_IMAGE - Presents GUI for reading image files.

DIALOG_WRITE_IMAGE - Presents GUI for writing image files.

IDLffDICOM - Contains the data for one or more images embedded in a DICOM part 10 file.

IDLffDicomEx - Allows you to read and write data contained in a DICOM file. See the *Medical Imaging in IDL* manual.

IDLffJPEG2000 - Contains the data for one or more JPEG2000 files.

IDLffMrSID - Allows you to query and load image data from a MrSID image file.

MPEG_CLOSE - Closes an MPEG sequence.

MPEG_OPEN - Opens an MPEG sequence.

MPEG_PUT - Inserts an image array into an MPEG sequence.

MPEG_SAVE - Saves an MPEG sequence to a file.

READ_BMP - Reads Microsoft Windows bitmap file (.BMP).

READ_DICOM - Reads an image from a DICOM file.

READ_GIF - Reads GIF file (.GIF)

READ_IMAGE - Reads the image contents of a file and returns the image in an IDL variable.

READ_INTERFILE - Reads Interfile (v3.3) file.

READ_JPEG - Reads JPEG file.

READ_JPEG2000 - Reads JPEG 2000 file.

READ_MRSID - Reads MrSID file.

READ_PICT - Reads Macintosh PICT (version 2) bitmap file.

READ_PNG - Reads Portable Network Graphics (PNG) file.

READ_PPM - Reads PGM (gray scale) or PPM (portable pixmap for color) file.

READ_SRF - Reads Sun Raster Format file.

READ_TIFF - Reads TIFF format file.

READ_X11_BITMAP - Reads X11 bitmap file.

READ_XWD - Reads X Windows Dump file.

TVRD - Reads an image from a window into a variable.

WRITE_BMP - Writes Microsoft Windows Version 3 device independent bitmap file (.BMP).

WRITE_GIF - Writes GIF file (.GIF).

WRITE_IMAGE - Writes an image and its color table vectors, if any, to a file of a specified type.

WRITE_JPEG - Writes JPEG file.

WRITE_JPEG2000 - Writes JPEG 2000 file.

WRITE_NRIF - Writes NCAR Raster Interchange Format rasterfile.

WRITE_PICT - Writes Macintosh PICT (version 2) bitmap file.

WRITE_PNG - Writes Portable Network Graphics (PNG) file.

WRITE_PPM - Writes PPM (true-color) or PGM (gray scale) file.

WRITE_SRF - Writes Sun Raster File (SRF).

WRITE_TIFF - Writes TIFF file with 1 to 3 channels.

XOBJVIEW_WRITE_IMAGE - Write the object currently displayed in XOBJVIEW to an image file.

Scientific Data Formats

CDF Routines - see the Scientific Data Formats chapter.

EOS Routines - see the Scientific Data Formats chapter.

H5_BROWSER - Opens a GUI to view contents of HDF5 files.

HDF Routines - see the Scientific Data Formats chapter.

HDF5 Routines - see the Scientific Data Formats chapter.

HDF_BROWSER - Opens a GUI to view contents of HDF, HDF-EOS, or NetCDF file.

HDF_READ - Extracts HDF, HDF-EOS, and NetCDF data and metadata into an output structure.

NCDF Routines - see the Scientific Data Formats chapter.

Other Data Formats

ASCII_TEMPLATE - Presents a GUI that generates a template defining an ASCII file format.

BINARY_TEMPLATE - Presents a GUI for interactively generating a template structure for use with READ_BINARY.

IDLffDXF - Object that contains geometry, connectivity, and attributes for graphics primitives.

IDLffShape - Contains geometry, connectivity and attributes for graphics primitives accessed from ESRI Shapefiles.

IDLffXMLDOM* - Represents classes that provide support for IDL's XML Document Object Model (DOM). See "IDLffXMLDOM Classes" on page 80.

IDLffXMLSAX - Represents an XML SAX level 2 parser.

IDLgrVRML - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.

READ_ASCII - Reads data from an ASCII file.

READ_BINARY - Reads the contents of a binary file using a passed template or basic command line keywords.

READ_SYLK - Reads Symbolic Link format spreadsheet file.

READ_WAV - Reads the audio stream from the named .WAV file.

READ_WAVE - Reads Wavefront Advanced Visualizer file.

WRITE_SYLK - Writes SYLK (Symbolic Link) spreadsheet file.

WRITE_WAV - Writes the audio stream to the named .WAV file.

WRITE_WAVE - Writes Wavefront Advanced Visualizer (.WAV) file.

General Input/Output

ASSOC - Associates an array structure with a file.

CLOSE - Closes the specified files.

COPY_LUN - Copies data between two open files.

EOF - Tests the specified file for the end-of-file condition.

FLUSH - Flushes file unit buffers.

FREE_LUN - Frees previously-reserved file units.

GET_KBRD - Gets one input IDL character.

GET_LUN - Reserves a logical unit number (file unit).

IOCTL - Performs special functions on UNIX files.

OPENR/OPENU/OPENW - Opens files for reading, updating, or writing.

POINT_LUN - Sets or gets current position of the file pointer.

PRINT/PRINTF - Writes formatted output to screen or file.

READ/READF - Reads formatted input from keyboard or file.

READS - Reads formatted input from a string variable.

READU - Reads unformatted binary data from a file.

SHMMAP - Maps anonymous shared memory, or local disk files, into the memory address space of the currently executing IDL process.

SHMUNMAP - Removes a memory segment previously created by SHMMAP from the system.

SHMVAR - Creates an IDL array variable that uses the memory from a current mapped memory segment created by the SHMMAP procedure.

SKIP_LUN - Reads data in an open file and moves the file pointer.

SOCKET - Opens a client-side TCP/IP Internet socket as an IDL file unit.

TRUNCATE_LUN - Truncates an open file at the location of the current file pointer.

WRITEU - Writes unformatted binary data to a file.

Language Catalogs

IDLffLangCat - Finds and loads an XML language catalog.

LOCALE_GET - Returns the current locale of the operating platform.

MULTI - Returns a catalog object for the given parameters if found.

Mapping

IDLffShape - Contains geometry, connectivity and attributes for graphics primitives accessed from ESRI Shapefiles.

IMAP - Displays georeferenced data in an iTool.

LL_ARC_DISTANCE - Returns the longitude and latitude of a point given arc distance and azimuth.

MAP_2POINTS - Returns distance, azimuth, and path relating to the great circle or rhumb line connecting two points on a sphere.

MAP_CONTINENTS - Draws continental boundaries, filled continents, political boundaries, coastlines, and/or rivers, over an existing map projection established by MAP_SET.

MAP_GRID - Draws parallels and meridians over a map projection.

MAP_IMAGE - Returns an image warped to fit the current map projection. (Use when map data is larger than the display).

MAP_PATCH - Returns an image warped to fit the current map projection. (Use when map data is smaller than the display).

MAP_PROJ_FORWARD - Transforms map coordinates from longitude/latitude to Cartesian (X, Y) coordinates

MAP_PROJ_IMAGE - Warps an image from geographic coordinates to a specified map projection.

MAP_PROJ_INFO - Returns information about current map and/or the available projections.

MAP_PROJ_INIT - Initializes a mapping projection, using either IDL's own map projections or the General Cartographic Transformation Package (GCTP) map projections.

MAP_PROJ_INVERSE - Transforms map coordinates from Cartesian (X, Y) coordinates to longitude/latitude.

MAP_SET - Establishes map projection type and limits.

Mathematics

Complex Numbers

COMPLEX - Converts argument to complex type.

CONJ - Returns the complex conjugate of X.

DCOMPLEX - Converts argument to double-precision complex type.

IMAGINARY - Returns the imaginary part of a complex value.

REAL_PART - Returns the real part of a complex-valued argument.

Correlation Analysis

A_CORRELATE - Computes autocorrelation.

C_CORRELATE - Computes cross correlation.

CORRELATE - Computes the linear Pearson correlation.

M_CORRELATE - Computes multiple correlation coefficient.

P_CORRELATE - Computes partial correlation coefficient.

R_CORRELATE - Computes rank correlation.

Curve and Surface Fitting

COMFIT - Fits paired data using one of six common filtering functions.

CRVLENGTH - Computes the length of a curve.

CURVEFIT - Fits multivariate data with a user-supplied function.

GAUSS2DFIT - Fits a 2D elliptical Gaussian equation to rectilinearly gridded data.

GAUSSFIT - Fits the sum of a Gaussian and a quadratic.

GRID_TPS - Uses thin plate splines to interpolate a set of values over a regular 2D grid, from irregularly sampled data values.

KRIG2D - Interpolates set of points using kriging.

LADFIT - Fits paired data using least absolute deviation method.

LINFIT - Fits by minimizing the Chi-square error statistic.

LMFIT - Does a non-linear least squares fit.

MIN_CURVE_SURF - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.

POLY_FIT - Performs a least-square polynomial fit.

REGRESS - Computes fit using multiple linear regression.

SFIT - Performs polynomial fit to a surface.

SVDFIT - Multivariate least squares fit using SVD method.

TRIGRID - Interpolates irregularly-gridded data to a regular grid from a triangulation.

Differentiation and Integration

CRVLENGTH - Computes the length of a curve.

DERIV - Performs differentiation using 3-point Langrangian interpolation.

DERIVSIG - Computes standard deviation of derivative found by DERIV.

INT_2D - Computes the double integral of a bivariate function.

INT_3D - Computes the triple integral of a trivariate function.

INT_TABULATED - Integrates a tabulated set of data.

LSODE - Advances a solution to a system of ordinary differential equations one time-step H.

QROMB - Evaluates integral over a closed interval.

QROMO - Evaluates integral over an open interval.

QSIMP - Evaluates integral using Simpson's rule.

RK4 - Solves differential equations using fourth-order Runge-Kutta method.

Eigenvalues and Eigenvectors

EIGENQL - Computes eigenvalues and eigenvectors of a real, symmetric array.

EIGENVEC - Computes eigenvectors of a real, non-symmetric array.

ELMHES - Reduces nonsymmetric array to upper Hessenberg form.

HQR - Returns all eigenvalues of an upper Hessenberg array.

TRIQL - Determines eigenvalues and eigenvectors of tridiagonal array.

TRIRED - Reduces a real, symmetric array to tridiagonal form.

Gridding and Interpolation

BILINEAR - Computes array using bilinear interpolation.

CONGRID - Shrinks or expands the size of an array by an arbitrary amount.

GRID_INPUT - Preprocesses and sorts two-dimensional scattered data points, and removes duplicate values.

GRID_TPS - Uses thin plate splines to interpolate a set of values over a regular 2D grid, from irregularly sampled data values.

GRID3 - Creates a regularly-gridded 3D dataset from a set of scattered 3D nodes.

GRIDDATA - Interpolates scattered data values and locations sampled on a plane or a sphere to a regular grid.

INTERPOL - Performs linear interpolation on vectors.

INTERPOLATE - Returns an array of interpolates.

KRIG2D - Interpolates set of points using kriging.

MIN_CURVE_SURF - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.

POLAR_SURFACE - Interpolates a surface from polar coordinates to rectangular coordinates.

SPH_SCAT - Performs spherical gridding.

SPL_INIT - Establishes the type of interpolating spline.

SPL_INTERP - Performs cubic spline interpolation (Numerical Recipes).

REBIN - Resizes a vector or an array to a set of given dimensions.

SPLINE - Performs cubic spline interpolation.

SPLINE_P - Performs parametric cubic spline interpolation.

TRI_SURF - Interpolates gridded set of points with a smooth quintic surface.

TRIANGULATE - Constructs Delaunay triangulation of a planar set of points.

TRIGRID - Interpolates irregularly-gridded data to a regular grid from a triangulation.

VALUE_LOCATE - Finds the intervals within a given monotonic vector that brackets a given set of one or more search values.

VORONOI - Computes Voronoi polygon given Delaunay triangulation.

Hypothesis Testing

CTI_TEST - Performs chi-square goodness-of-fit test.

FV_TEST - Performs the F-variance test.

KW_TEST - Performs Kruskal-Wallis H-test.

LNP_TEST - Computes the Lomb Normalized Periodogram.

MD_TEST - Performs the Median Delta test.

R_TEST - Runs test for randomness.

RS_TEST - Performs the Wilcoxon Rank-Sum test.

S_TEST - Performs the Sign test.

TM_TEST - Performs t-means test.

XSQ_TEST - Computes Chi-square goodness-of-fit test.

LAPACK Routines

LA_CHOLDC - Computes the Cholesky factorization of an *n*-by-*n* symmetric positive-definite array.

LA_CHOLMPROVE - Uses Cholesky factorization to improve the solution to a system of linear equations.

LA_CHOLSOL - Used in conjunction with LA_CHOLDC to solve a set of linear equations.

LA_DETERM - Uses LU decomposition to compute the determinant of a square array.

LA_EIGENPROBLEM - Uses the QR algorithm to compute eigenvalues and eigenvectors of an array.

LA_EIGENQL - Computes selected eigenvalues and eigenvectors.

LA_EIGENVEC - Uses the QR algorithm to compute all of some eigenvectors of an array.

LA_ELMHES - Reduces a real nonsymmetric or complex array to upper Hessenberg from.

LA_GM_LINEAR_MODEL - Used to solve a general Gauss-Markov linear model problem.

LA_HQR - Uses the multishift QR algorithm to compute all eigenvalues of an array.

LA_INVERT - Uses LU decomposition to compute the inverse of a square array.

LA_LEAST_SQUARE_EQUALITY - Used to solve linear least-squares problems.

LA_LEAST_SQUARES - Used to solve linear least-squares problems.

LA_LINEAR_EQUATION - Uses LU decomposition to sole a system of linear equations.

LA_LUDC - Computes the LU decomposition of an array.

LA_LUMPROVE - Uses LU decomposition to improve the solution to a system of linear equations.

LA_LUSOL - Used in conjunction with LA_LUDC to solve a set of linear equations.

LA_SVD - Computes the singular value decomposition of an array.

LA_TRIDC - Computes the LU decomposition of a tridiagonal array.

LA_TRIMPROVE - Improves the solution to a system of linear equations with a tridiagonal array.

LA_TRIQL - Uses the QL and QR variants of the implicitly-shifted QR algorithm to compute the eigenvalues and eigenvectors of an array.

LA_TRIRED - Reduces a real symmetric or complex Hermitian array to real tridiagonal from.

LA_TRISOL - Used in conjunction with LA_TRIDC to solve a set of linear equations.

Linear Systems

CHOLDC - Constructs Cholesky decomposition of a matrix.

CHOLSOL - Solves set of linear equations (use with CHOLDC).

COND - Computes the condition number of a square matrix.

CRAMER - Solves system of linear equations using Cramer's rule.

CROSSP - Computes vector cross product.

DETERM - Computes the determinant of a square matrix.

GS_ITER - Solves linear system using Gauss-Seidel iteration.

IDENTITY - Returns an identity array.

INVERT - Computes the inverse of a square array.

LINBCG - Solves a set of sparse linear equations using the iterative biconjugate gradient method.

LU_COMPLEX - Solves complex linear system using LU decomposition.

LUDC - Replaces array with the LU decomposition.

LUMPROVE - Uses LU decomposition to iteratively improve an approximate solution.

LUSOL - Solves a set of linear equations. Use with LUDC.

NORM - Computes Euclidean norm of vector or Infinity norm of array.

SVDC - Computes Singular Value Decomposition of an array.

SVSOL - Solves set of linear equations using back-substitution.

TRACE - Computes the trace of an array.

TRISOL - Solves tridiagonal systems of linear equations.

Mathematical Error Assessment

CHECK_MATH - Returns and clears accumulated math error status.

FINITE - Returns True if its argument is finite.

MACHAR - Determines and returns machine-specific parameters affecting floating-point arithmetic.

Miscellaneous Math Routines

ABS - Returns the absolute value of X.

CEIL - Returns the closest integer greater than or equal to X.

CIR_3PNT - Returns radius and center of circle, given 3 points.

COMPLEXROUND - Rounds a complex array.

DIAG_MATRIX - Constructs a diagonal matrix from an input vector, or if given a matrix, then extracts a diagonal vector.

DIST - Creates array with each element proportional to its frequency.

EXP - Returns the natural exponential function of given expression.

FLOOR - Returns closest integer less than or equal to argument.

IMAGINARY - Returns the imaginary part of a complex value.

ISHFT - Performs integer bit shift.

LEEFILT - Performs the Lee filter algorithm on an image array.

MATRIX_MULTIPLY - Calculates the IDL matrix-multiply operator (#) of two (possibly transposed) arrays.

MATRIX_POWER - Computes the product of a matrix with itself.

PNT_LINE - Returns the perpendicular distance between a point and a line.

POLY_AREA - Returns the area of a polygon given the coordinates of its vertices.

PRIMES - Computes the first K prime numbers.

PRODUCT - Returns the product of elements within an army.

ROUND - Returns the integer closest to its argument.

SPH_4PNT - Returns center and radius of a sphere given 4 points.

SQRT - Returns the square root of X.

TOTAL - Sums of the elements of an array.

VOIGT - Calculates intensity of atomic absorption line (Voight) profile.

Multivariate Analysis

CLUST_WTS - Computes cluster weights of array for cluster analysis.

CLUSTER - Performs cluster analysis.

CLUSTER_TREE - Computes the hierarchical clustering for a set of *m* items in an *n*-dimensional space.

CTI_TEST - Performs chi-square goodness-of-fit test.

DENDRO_PLOT - Draws a two-dimensional dendrite plot on the current direct graphics device if given a hierarchical tree cluster, as created by CLUSTER_TREE.

DENDROGRAM - Constructs a dendrogram and returns a set of vertices and connectivity that can be used to visualize the dendrite plot if given a hierarchical tree cluster, as created by CLUSTER_TREE.

DISTANCE_MEASURE - Computes the pairwise distance between a set of items or observations.

KW_TEST - Performs Kruskal-Wallis H-test.

M_CORRELATE - Computes multiple correlation coefficient.

P_CORRELATE - Computes partial correlation coefficient.

PCOMP - Computes principal components/derived variables.

STANDARDIZE - Computes standardized variables.

Nonlinear Equations

BROYDEN - Solves nonlinear equations using Broyden's method.

FX_ROOT - Computes real and complex roots of a univariate nonlinear function using an optimal Müller's method.

FZ_ROOTS - Finds the roots of a complex polynomial using Laguerre's method.

NEWTON - Solves nonlinear equations using Newton's method.

Optimization

AMOEBA - Minimizes a function using downhill simplex method.

CONSTRAINED_MIN - Minimizes a function using Generalized Reduced Gradient Method.

DFPMIN - Minimizes a function using Davidon-Fletcher-Powell method.

POWELL - Minimizes a function using the Powell method.

SIMPLEX - Use the simplex method to solve linear programming problems.

Probability

BINOMIAL - Computes binomial distribution function.

CHISQR_CVF - Computes cutoff value in a Chi-square distribution.

CHISQR_PDF - Computes Chi-square distribution function.

F_CVF - Computes the cutoff value in an F distribution.

F_PDF - Computes the F distribution function.

GAUSS_CVF - Computes cutoff value in Gaussian distribution.

GAUSS_PDF - Computes Gaussian distribution function.

GAUSSINT - Returns integral of Gaussian probability function.

T_CVF - Computes the cutoff value in a Student's t distribution.

T_PDF - Computes Student's t distribution.

Sparse Arrays

NOTE: SPRSIN must be used to convert to sparse storage format before the other routines can be used.

FULSTR - Restores a sparse matrix to full storage mode.

LINBCG - Solves a set of sparse linear equations using the iterative biconjugate gradient method.

READ_SPR - Reads a row-indexed sparse matrix from a file.

SPRSAB - Performs matrix multiplication on sparse matrices.

SPRSAX - Multiplies sparse matrix by a vector.

SPRSIN - Converts matrix to row-index sparse matrix.

SPRSTP - Constructs the transpose of a sparse matrix.

WRITE_SPR - Writes row-indexed sparse array structure to a file.

Special Math Functions

BESELI - Returns the I Bessel function of order N for X.

BESELJ - Returns the J Bessel function of order N for X.

BESELK - Returns the K Bessel function of order N for X.

BESELY - Returns the Y Bessel function of order N for X.

BETA - Returns the value of the beta function.

ERF - Returns the value of an error function.

ERFC - Returns the value of a complementary error function.

ERFCX - Returns the value of a scaled complementary error function.

EXPINT - Returns the value of the exponential integral.

GAMMA - Returns the gamma function of X.

IBETA - Computes the incomplete beta function.

IGAMMA - Computes the incomplete gamma function.

LAGUERRE - Returns value of the associated Laguerre polynomial.

LEGENDRE - Returns value of the associated Legendre polynomial.

LNGAMMA - Returns logarithm of the gamma function of X.

POLY - Evaluates polynomial function of a variable.

SPHER_HARM - Returns value of the spherical harmonic function.

Statistical Fitting

COMFIT - Fits paired data using one of six common filtering functions.

CURVEFIT - Fits multivariate data with a user-supplied function.

FUNCT - Evaluates the sum of a Gaussian and a 2nd-order polynomial and optionally returns the value of its partial derivatives.

LADFIT - Fits paired data using least absolute deviation method.

LINFIT - Fits by minimizing the Chi-square error statistic.

REGRESS - Multiple linear regression.

SVDFIT - Multivariate least squares fit using SVD method.

Statistical Tools

FACTORIAL - Computes the factorial N!.

HIST_2D - Returns histogram of two variables.

HISTOGRAM - Computes the density function of an array.

KURTOSIS - Computes statistical kurtosis of n-element vector.

MAX - Returns the value of the largest element of an array.

MEAN - Computes the mean of a numeric vector.

MEANABSDEV - Computes the mean absolute deviation of a vector.

MEDIAN - Returns the median value of Array or applies a median filter.

MIN - Returns the value of the smallest element of an array.

MOMENT - Computes mean, variance, skewness, and kurtosis.

RANDOMN - Returns normally-distributed pseudo-random numbers.

RANDOMU - Returns uniformly-distributed pseudo-random numbers.

RANKS - Computes magnitude-based ranks.

SKEWNESS - Computes statistical skewness of an n-element vector.

SORT - Returns the indices of an array sorted in ascending order.

STDDEV - Computes the standard deviation of an n-element vector.

TOTAL - Sums of the elements of an array.

VARIANCE - Computes the statistical variance of an n-element vector.

Time-Series Analysis

A_CORRELATE - Computes autocorrelation.

C_CORRELATE - Computes cross correlation.

SMOOTH - Smooths with a boxcar average.

TS_COEF - Computes the coefficients for autoregressive time-series.

TS_DIFF - Computes the forward differences of a time-series.

TS_FCAST - Computes future or past values of stationary time-series.

TS_SMOOTH - Computes moving averages of a time-series.

Transcendental Functions

ACOS - Returns the arc-cosine of X.

ALOG - Returns the natural logarithm of X.

ALOG10 - Returns the logarithm to the base 10 of X.

ASIN - Returns the arc-sine of X.

ATAN - Returns the arc-tangent of X.

COS - Returns the cosine of X.

COSH - Returns the hyperbolic cosine of X.

EXP - Returns the natural exponential function of a given expression.

SIN - Returns the trigonometric sine of X.

SINH - Returns the hyperbolic sine of X.

TAN - Returns the tangent of X.

TANH - Returns the hyperbolic tangent of X.

Transforms

BLK_CON - Convolves input signal with impulse-response sequence.

CHEBYSHEV - Returns the forward or reverse Chebyshev polynomial expansion.

CONVOL - Convolves two vectors or arrays.

FFT - Returns the Fast Fourier Transform of an array.

HILBERT - Constructs a Hilbert transform.

HOUGH - Returns the Hough transform of a two-dimensional image.

RADON - Returns the Radon transform of a two-dimensional image.

WTN - Returns wavelet transform of the input array.

See Also - Wavelet Toolkit

Object Class Library

Analysis Objects

IDLanROI - Represents a region of interest.

IDLanROIGroup - Analytical representation of a group of regions of interest.

File Format Objects

IDLffDICOM - Contains the data for one or more images embedded in a DICOM Part 10 file.

IDLffDicomEx - Allows you to read and write data contained in a DICOM file. See the *Medical Imaging in IDL* manual.

IDLffDXF - Contains geometry, connectivity and attributes for graphics primitives.

IDLffJPEG2000 - Contains the data for one or more images embedded in a JPEG-2000 file as well as functionality for reading and writing JPEG-2000 files.

IDLffLangCat - Finds and loads an XML language catalog.

IDLffMrSID - Used to query information about and load image data from a MrSID (.sid) image file.

IDLffShape - Contains geometry, connectivity and attributes for graphics primitives.

IDLffXMLDOMAttr - Represents an attribute that is part of an element object in an XML document.

IDLffXMLDOMCDATASection - Used to escape blocks of text in an XML document containing characters that would otherwise be regarded as markup.

IDLffXMLDOMCharacterData - Extension of the IDLffXMLDOMNode class that supplies a set of methods for accessing character data in the XML DOM tree.

IDLffXMLDOMComment - Represents the content of an XML comment (characters between "<!--" and "-->").

IDLffXMLDOMDocument - Represents the entire XML document as the root of the XML document tree and by providing the primary access to the document's data.

IDLffXMLDOMDocumentFragment - Represents a document fragment in an XML document.

IDLffXMLDOMDocumentType - References a DocumentType node in an XML document.

IDLffXMLDOMElement - References an element node in an XML document.

IDLffXMLDOMEntity - References an entity, either parsed or unparsed, in an XML document.

IDLffXMLDOMEntityReference - References an entity reference node in an XML document.

IDLffXMLDOMNamedNodeMap - Container for IDLffXMLDOM nodes that uses node names to access the nodes.

IDLffXMLDOMNode - Abstract class used as a superclass for other IDLffXMLDOM node classes.

IDLffXMLDOMNodeIterator - Allows iterative navigation of the XML DOM tree.

IDLffXMLDOMNodeList - Container for IDLffXMLDOM nodes.

IDLffXMLDOMNotation - Represents a notation in the XML DTD.

IDLffXMLDOMProcessingInstruction - Represents a processing instruction in an XML document.

IDLffXMLDOMText - References a text node in the XML document.

IDLffXMLDOMTreeWalker - Allows tree-walking navigation of the XML DOM tree.

IDLffXMLSAX - Represents an XML SAX level 2 parser.

IDLgrMPEG - Creates an MPEG movie file from an array of image frames.

IDLgrVRML - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.

Graphic Objects—Destination

IDL_Container - A container for other objects.

IDLgrBuffer - An in-memory, off-screen destination object.

IDLgrClipboard - A destination object representing the native clipboard.

IDLgrPrinter - Represents a hardcopy graphics destination.

IDLgrWindow - Represents an on-screen area on a display device that serves as a graphics destination.

Graphic Objects—Display

IDL_Container - Object that holds other objects.

IDLgrModel - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated).

IDLgrScene - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects.

IDLgrView - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class.

IDLgrViewgroup - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects.

Graphic Objects—Visualization

IDLgrAxis - Represents a single vector that may include a set of tick marks, tick labels, and a title.

IDLgrColorbar - Consists of a color-ramp with an optional framing box and annotation axis.

IDLgrContour - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points.

IDLgrFont - Represents a typeface, style, weight, and point size that may be associated with text objects.

IDLgrImage - Represents a mapping from a 2D array of data values to a 2D array of pixel colors, resulting in a flat 2D-scaled version of the image, drawn at Z = 0.

IDLgrLegend - Provides a simple interface for displaying a legend.

IDLgrLight - Represents a source of illumination for 3D graphic objects.

IDLgrPalette - Represents a color lookup table that maps indices to red, green, and blue values.

IDLgrPattern - Describes which pixels are filled and which are left blank when an area is filled.

IDLgrPlot - Creates set of polylines connecting data points in 2D space.

IDLgrPolygon - Represents one or more polygons that share a set of vertices and rendering attributes.

IDLgrPolyline - Represents one or more polylines that share a set of vertices and rendering attributes.

IDLgrROI - Object graphics representation of a region of interest.

IDLgrROIGroup - Object Graphics representation of a group of regions of interest.

IDLgrSurface - A shaded or vector representation of a mesh grid. No superclasses.

IDLgrSymbol - Represents a graphical element that is plotted relative to a particular position.

IDLgrTessellator - Converts a simple concave polygon (or a simple polygon with "holes") into a number of simple convex polygons (general triangles).

IDLgrText - Represents one or more text strings that share common rendering attributes.

IDLgrView - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class.

IDLgrViewgroup - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects.

IDLgrVolume - Represents mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions.

iTools System Objects

IDLitCommand - The base functionality for the iTools command buffer system.

IDLitCommandSet - A container for IDLitCommand objects, which allows a group of commands to be managed as a single item.

IDLitComponent - A core or base component, from which all other components subclass.

IDLitContainer - A specialization of the IDL_Container class that manages a collection of IDLitComponents and provides methods for working with the Identifier system of the iTools framework.

IDLitData - A generic data storage object that can hold any IDL data type available. It provides typing, metadata, and data change notification functionality. When coupled with IDLitDataContainer, it forms the element for the construction of composite data types.

IDLitDataContainer - A container for IDLitData and IDLitDataContainer objects. This container is used to form hierarchical data structures. Data and DataContainer objects can be added and removed to/from a DataContainer during program execution, allowing for dynamic creation of composite data types.

IDLitDataOperation - A subclass to IDLitOperation that automates data access and automatically records information for the undo-redo system.

IDLitIMessaging - An interface providing common methods to send or trigger messaging and error actions, which may occur during execution.

IDLitManipulator - The base functionality of the iTools manipulator system.

IDLitManipulatorContainer - A container for IDLitManipulator objects, which allows for the construction of manipulator hierarchies. This container implements the concept of a current manipulator for the items it contains.

IDLitManipulatorManager - A specialization of the manipulator container (IDLitManipulatorContainer), which acts as the root of the manipulator hierarchy.

IDLitManipulatorVisual - The means for iTool developers to create visual elements associated with an interactive manipulator.

IDLitOperation - The basis for all iTool operations. It defines how an operation is executed and how information about the operation is recorded for the command transaction (undo-redo) system.

IDLitParameter - An interface providing parameter management methods to associate parameter names with IDLitData objects.

IDLitParameterSet - A specialized subclass of the IDLitDataContainer class. This class provides the ability to associate names with contained IDLitData objects.

IDLitReader - The definition of the interface and the process used to construct file readers for the iTools framework. When a new file reader is constructed for the iTools system, a new class is subclassed from this IDLitReader class.

IDLitTool - All the functionality provided by a particular instance of an IDL Intelligent Tool (iTool). This object provides the management systems for the underlying tool functionality.

IDLitUI - A link between the underlying functionality of an iTool and the IDL widget interface.

IDLitVisualization - The basis for all iTool visualizations. All visualization components subclass from this class.

IDLitWindow - The basis for all iTool windows. All iTool windows subclass from this class.

IDLitWriter - The definition of the interface and the process used to construct file writers for the iTools framework. When a new file writer is constructed for the iTools system, a new class is subclassed from this IDLitWriter class.

Miscellaneous Objects

IDL_Container - A container for other objects.

IDL_Savefile - Provides complete query and restore capabilities for IDL SAVE files.

IDLcomActiveX - Creates an IDL object that encapsulates an ActiveX control.

IDLcomIDispatch - Creates an IDL object that encapsulates a COM object.

IDLjavaObject - An IDL object encapsulating a Java object. IDL provides data type and other translation services, allowing IDL programs to access the Java object's methods and properties using standard IDL syntax.

TrackBall - Translates widget events from a draw widget into transformations that emulate a virtual trackball (for transforming object graphics in three dimensions).

Operating System Access

APP_USER_DIR - Provides access to the *application user directory.*

APP_USER_DIR_QUERY - Locates existing *application user directories.*

CALL_EXTERNAL - Calls a function in an external sharable object and returns a scalar value.

CD - Sets and/or changes the current working directory.

FILE_BASENAME - Returns the *basename* of a *file path*.

FILE_CHMOD - Changes file access permissions.

FILE_DELETE - Deletes files and empty directories.

FILE_DIRNAME - Returns the *dirname* of a *file path*.

FILE_EXPAND_PATH - Fully qualifies file and directory paths.

FILE_INFO - Returns status information about a file.

FILE_MKDIR - Creates directories.

FILE_SAME - Determines whether two different file names refer to the same underlying file.

FILE_SEARCH - Returns a string array containing the names of all files matching the input path specification.

FILE_TEST - Test a file or directory for existence and other specific attributes.

FILE_WHICH - Searches for a specified file in a directory search path.

GET_DRIVE_LIST (Windows only) - Returns string array of the names of valid drives/volumes for the file system.

GET_SCREEN_SIZE - Returns dimensions of the screen.

GETENV - Returns the value of an environment variable.

LINKIMAGE - Merges routines written in other languages with IDL at run-time.

PATH_SEP - Returns the proper file path segment separator character for the current operating system.

POPD - Removes the top directory on the working directory stack maintained by PUSHD/POPD.

PRINTD - Prints contents of the directory stack maintained by PUSHD/POPD.

PUSHD - Pushes a directory to top of directory stack maintained by PUSHD/POPD.

SETENV - Adds or changes an environment variable.

SPAWN - Spawns child process for access to operating system.

Performance Testing

MEMORY - Provides information about the amount of dynamic memory currently in use by the IDL session.

PROFILER - Accesses the IDL Code Profiler used to analyze performance of applications.

TIME_TEST2 - Performs speed benchmarks for IDL.

Plotting

AXIS - Draws an axis of the specified type and scale.

BAR_PLOT - Creates a bar graph.

ERRPLOT - Plots error bars over a previously drawn plot.

IDLgrAxis - Displays a plot axis object that may include tick marks, text and a title.

IDLgrContour - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points.

IDLgrLegend - Provides a simple interface for displaying a legend.

IDLgrPlot - Creates a set of polylines connecting data points in two-dimensional space.

IDLgrSymbol - Represents a graphical element that is plotted relative to a particular position.

IPLOT - Creates an iTool and associated user interface (UI) configured to display and manipulate plot data.

LABEL_DATE - Labels axes with dates. Use with [XYZ]TICKFORMAT keyword.

OPLOT - Plots vector data over a previously-drawn plot.

OPLOTERR - Draws error bars over a previously drawn plot.

PLOT - Plots vector arguments as X versus Y graphs.

PLOT_3DBOX - Plots function of two variables inside 3D box.

PLOT_FIELD - Plots a 2D field using arrows.

PLOTERR - Plots individual data points with error bars.

PLOTS - Plots vectors and points.

POLYFILL - Fills the interior of a polygon.

POLYFILLV - Returns subscripts of pixels inside a polygon.

PROFILE - Extracts a profile from an image.

PROFILES - Interactively examines image profiles.

THREED - Plots a 2D array as a pseudo 3D plot.

TRIANGULATE - Constructs Delaunay triangulation of a planar set of points.

TRIGRID - Interpolates irregularly-gridded data to a regular grid from a triangulation.

USERSYM - Defines a new plotting symbol.

VEL - Draws a velocity (flow) field with streamlines.

VELOVECT - Draws a 2D velocity field plot.

WF_DRAW - Draws weather fronts with smoothing.

XPLOT3D - Utility for creating and interactively manipulating 3D plots.

XYOUTS - Draws text on currently-selected graphics device.

Programming and IDL Control

APP_USER_DIR - Provides access to the *application user directory.*

APP_USER_DIR_QUERY - Locates existing *application user directories.*

ARG_PRESENT - Returns TRUE if the value of the specified variable can be passed back to the caller.

BIT_FFS - Returns the index of the first bit set (non-zero) in an integer.

BIT_POPULATION - Returns the number of set (non-zero) bits in an integer.

BREAKPOINT - Sets and clears breakpoints for debugging.

BYTEORDER - Converts between host and network byte ordering.

CALL_FUNCTION - Calls an IDL function.

CALL_METHOD - Calls an IDL object method.

CALL_PROCEDURE - Calls an IDL procedure.

CATCH - Declares and clears exception handlers.

COMMAND_LINE_ARGS - Returns string values supplied when the user starts IDL with the -arg or -args command line options.

CPU - Changes the values stored in the read-only !CPU system variable.

CREATE_STRUCT - Creates and concatenates structures.

COMPILE_OPT - Change default rules for compiling routines.

DEFINE_KEY - Programs keyboard function keys.

DEFINE_MSGBLK - Defines and loads a new message block into the current IDL session.

DEFINE_MSGBLK_FROM_FILE - Reads the definition of a message block from a file, and loads it into the current IDL session.

DEFSYSV - Creates a new system variable.

EXECUTE - Compiles, executes IDL statements contained in a string.

EXIT - Quits IDL and exits back to the operating system.

EXPAND_PATH - Expands path-definition string into full path name for use with the !PATH system variable.

HEAP_FREE - Recursively frees all heap variables referenced by its input argument.

HEAP_GC - Performs “garbage collection” on heap variables.

HEAP_NOSAVE - Used to clear the *save attribute* of pointer or object heap variables.

HEAP_SAVE - Used to query whether a pointer or object heap variable is savable. It can also be used to change the heap variable save attribute.

IDL_VALIDNAME - Determines whether a string may be used as a valid IDL variable name or structure tag name.

IDLITSYS_CREATETOOL - Creates an instance of the specified tool registered within the iTools system.

ITCURRENT - Set the current tool in the iTools system.

ITDELETE - Deletes a tool in the iTools system.

ITGETCURRENT - Gets the identifier of the current tool in the iTools system.

ITREGISTER - Registers tool object classes with the iTools system.

ITRESET - Resets the iTools session.

ITRESOLVE - Resolves all IDL code within the iTools directory, as well as all other IDL code required for the iTools framework.

KEYWORD_SET - Returns True if given expression is defined and nonzero or an array.

LMGR - Determines the type of license used by the current IDL session.

LOGICAL_AND - Performs a logical AND operation on its arguments.

LOGICAL_OR - Performs a logical OR operation on its arguments.

LOGICAL_TRUE - Determines whether its arguments are non-zero (or non-NULL).

MESSAGE - Issues error and informational messages.

N_ELEMENTS - Returns the number of elements contained in an expression or variable.

N_PARAMS - Returns the number of non-keyword parameters used in calling an IDL procedure or function.

N_TAGS - Returns the number of tags in a structure.

OBJ_CLASS - Determines the class name of an object.

OBJ_DESTROY - Destroys an object reference.

OBJ_ISA - Determines inheritance relationship of an object.

OBJ_NEW - Creates an object reference.

OBJ_VALID - Verifies validity of object references.

ON_ERROR - Designates the error recovery method.

ON_IOERROR - Declares I/O error exception handler.

PATH_CACHE - Controls IDL's path caching mechanism.

PREF_COMMIT - Commits IDL preferences in the pending state.

PREF_GET - Returns information about IDL preferences.

PREF_MIGRATE - Imports IDL preferences from other versions of IDL for use by the currently running version.

PREF_SET - Sets new values for IDL preferences.

PTR_FREE - Destroys a pointer.

PTR_NEW - Creates a pointer.

PTR_VALID - Verifies the validity of pointers.

PTRARR - Creates an array of pointers.

RECALL_COMMANDS - Returns entries in IDL's command recall buffer.

REGISTER_CURSOR - Associates a given name with cursor information, used in conjunction with IDLgrWindow::SetCurrentCursor.

RESOLVE_ALL - Compiles any uncompiled routines.

RESOLVE_ROUTINE - Compiles a routine.

RETALL - Returns control to the main program level.

RETURN - Returns control to the next-higher program level.

ROUTINE_INFO - Provides information about compiled procedures and functions.

STOP - Stops the execution of a running program or batch file.

STRMESSAGE - Returns the text of a given error number.

STRUCT_ASSIGN - Uses "Relaxed Structure Assignment" to copy structures.

STRUCT_HIDE - Prevents the IDL HELP procedure from displaying information about structures or objects.

SWAP_ENDIAN - Reverses the byte ordering of scalars, arrays or structures.

SWAP_ENDIAN_INPLACE - Reverses the byte ordering of scalars, arrays or structures. Differs from the SWAP_ENDIAN function in that it alters the input data in place rather than making a copy.

TAG_NAMES - Returns the names of tags in a structure.

TEMPORARY - Returns a temporary copy of a variable, and sets the original variable to "undefined".

WAIT - Suspends execution of an IDL program for a specified period.

Query Routines

QUERY_ASCII - Obtains information about an ASCII file.

QUERY_BMP - Obtains information about a BMP image file.

QUERY_DICOM - Tests file for compatibility with READ_DICOM.

QUERY_GIF - Obtains information about a GIF image file.

QUERY_IMAGE - Determines if a file is recognized as an image file.

QUERY_JPEG - Obtains information about a JPEG image file.

QUERY_JPEG2000 - Obtains information about a JPEG 2000 image file.

QUERY_MRSID - Obtains information about a MrSID image file.

QUERY_PICT - Obtains information about a PICT image file.

QUERY_PNG - Obtains information about a PNG image file.

QUERY_PPM - Obtains information about a PPM image file.

QUERY_SRF - Obtains information about an SRF image file.

QUERY_TIFF - Obtains information about a TIFF image file.

QUERY_WAV - Checks that the file is actually a .WAV file and that the READ_WAV function can read the data in the file.

Saving/Restoring a Session

HEAP_NOSAVE - Used to clear the *save attribute* of pointer or object heap variables.

HEAP_SAVE - Used to query whether a pointer or object heap variable is savable. It can also be used to change the heap variable save attribute.

IDL_Savefile - Object that provides complete query and restore capabilities for IDL SAVE files.

JOURNAL - Logs IDL commands to a file.IDL.

RESTORE - Restores IDL variables and routines in an IDL SAVE file.

SAVE - Saves variables, system variables, and IDL routines in a file for later use.

Scientific Data Formats

CDF Routines - see the Scientific Data Formats chapter.

EOS Routines - see the Scientific Data Formats chapter.

H5_BROWSER - Opens a GUI to view contents of HDF5 files.

HDF Routines - see the Scientific Data Formats chapter.

HDF5 Routines - see the Scientific Data Formats chapter.

HDF_BROWSER - Opens a GUI to view contents of HDF, HDF-EOS, or NetCDF file.

HDF_READ - Extracts HDF, HDF-EOS, and NetCDF data and metadata into an output structure.

NCDF Routines - see the Scientific Data Formats chapter.

Scope Functions

SCOPE_LEVEL - Returns the current routine's scope level.

SCOPE_TRACEBACK - Returns the current interpreter call stack (the sequence of routine calls to the present point).

SCOPE_VARFETCH - Returns variables outside the current routine's local scope.

SCOPE_VARNAME - Returns the names of variables outside current routine's local scope.

Signal Processing

A_CORRELATE - Computes autocorrelation.

BLK_CON - Convolves input signal with impulse-response sequence.

C_CORRELATE - Computes cross correlation.

CONVOL - Convolves two vectors or arrays.

CORRELATE - Computes the linear Pearson correlation.

DIGITAL_FILTER - Calculates coefficients of a non-recursive, digital filter.

FFT - Returns the Fast Fourier Transform of an array.

HANNING - Creates Hanning and Hamming windows.

HILBERT - Constructs a Hilbert transform.

INTERPOL - Performs linear interpolation on vectors.

LEEFILT - Performs the Lee filter algorithm on an image array.

M_CORRELATE - Computes multiple correlation coefficient.

MEDIAN - Returns median value of an array or applies a median filter.

P_CORRELATE - Computes partial correlation coefficient.

R_CORRELATE - Computes rank correlation.

SAVGOL - Returns coefficients of Savitzky-Golay smoothing filter.

SMOOTH - Smooths with a boxcar average.

TS_COEF - Computes the coefficients for autoregressive time-series.

TS_DIFF - Computes the forward differences of a time-series.

TS_FCAST - Computes future or past values of stationary time-series.

TS_SMOOTH - Computes moving averages of a time-series.

WTN - Returns wavelet transform of the input array.

See Also - Wavelet Toolkit

Statements

Compound Statements

BEGIN...END - Defines a block of statements.

Conditional Statements

IF...THEN...ELSE - Conditionally executes a statement or block of statements.

CASE - Selects one statement for execution, depending on the value of an expression.

SWITCH - Selects one statement for execution, depending upon the value of an expression.

Loop Statements

FOR - Executes one or more statements repeatedly, incrementing or decrementing a variable with each repetition, until a condition is met.

REPEAT...UNTIL - Repeats statement(s) until expression evaluates to true. Subject is always executed at least once.

WHILE...DO - Performs statement(s) as long as expression evaluates to true. Subject is never executed if condition is initially false.

Jump Statements

BREAK - Exits from a loop (FOR, WHILE, REPEAT), CASE, or SWITCH statement.

CONTINUE - Starts the next iteration of the enclosing FOR, WHILE, or REPEAT loop.

GOTO - Transfers program control to point specified by *label*.

Functions and Procedures

COMPILE_OPT - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

FORWARD_FUNCTION - Causes argument(s) to be interpreted as functions rather than variables (versions of IDL prior to 5.0 used parentheses to declare arrays).

FUNCTION - Defines a function.

PRO - Defines a procedure.

Variable Scope

COMMON - Creates a common block.

String Processing

FILE_BASENAME - Returns the *basename* of a *file path*.

FILE_DIRNAME - Returns the *dirname* of a *file path*.

STRCMP - Compares two strings.

STRCOMPRESS - Removes whitespace from a string.

STREGEX - Performs regular expression matching.

STRING - Converts arguments to string type.

STRJOIN - Collapses a string scalar or array into merged strings.

STRLEN - Returns the length of a string.

STRLOWCASE - Converts a string to lower case.

STRMATCH - Compares search string against input string expression.

STRMID - Extracts a substring from a string.

STRPOS - Finds first occurrence of a substring within a string.

STRPUT - Inserts the contents of one string into another.

STRSPLIT - Splits its input string argument into separate substrings, according to the specified pattern.

STRTRIM - Removes leading and/or trailing blanks from string.

STRUPCASE - Converts a string to upper case.

Structures

IDL_VALIDNAME - Determines whether a string may be used as a valid IDL variable name or structure tag name.

N_TAGS - Returns the number of tags in a structure.

REPLICATE - Creates an array of given dimensions, filled with specified value.

STRUCT_ASSIGN - Uses “Relaxed Structure Assignment” to copy structures.

STRUCT_HIDE - Prevents the IDL HELP procedure from displaying information about structures or objects.

TAG_NAMES - Returns the names of tags in a structure.

Type Conversion

BYTE - Converts argument to byte type.

COMPLEX - Converts argument to complex type.

DCOMPLEX - Converts argument to double-precision complex type.

DOUBLE - Converts argument to double-precision type.

FIX - Converts argument to integer type, or type specified by TYPE keyword.

FLOAT - Converts argument to single-precision floating-point.

LONG - Converts argument to longword integer type.

LONG64 - Converts argument to 64-bit integer type.

STRING - Converts argument to string type.

UINT - Converts argument to unsigned integer type.

ULONG - Converts argument to unsigned longword integer type.

ULONG64 - Converts argument to unsigned 64-bit integer type.

Utilities

EFONT - Interactive vector font editor and display tool.

SLIDE_IMAGE - Creates a scrolling graphics window for examining large images.

XBM_EDIT - Creates, edits bitmap icons for IDL widget button labels.

XDISPLAYFILE - Displays ASCII text file in scrolling text widget.

XDXF - Utility to display and interactively manipulate DXF objects.

XFONT - Creates modal widget to select and view an X Windows font.

XINTERANIMATE - Displays animated sequence of images.

XLOADCT - Displays a tool for selecting and setting the current color table.

XMTOOL - Displays a tool for viewing XMANAGER widgets.

XOBJVIEW - Displays object viewer widget.

XOBJVIEW_ROTATE - Programmatically rotate the object currently displayed in XOBJVIEW.

XOBJVIEW_WRITE_IMAGE - Write the object currently displayed in XOBJVIEW to an image file.

XPALETTE - Displays a tool for creating and modifying color tables.

XPCOLOR - Adjusts the value of the current foreground plotting color, !P.COLOR.

XPLOT3D - Utility for creating and interactively manipulating 3D plots.

XROI - Utility for interactively defining and obtaining information about regions of interest.

XSURFACE - Provides a graphical interface to the SURFACE and SHADE_SURF commands.

XVAREDIT - Utility for editing any IDL variable.

XVOLUME - Utility for viewing and interactively manipulating volumes and isosurfaces.

XVOLUME_ROTATE - Utility for rotating a volume displayed in XVOLUME.

XVOLUME_WRITE_IMAGE - Utility for writing a volume displayed in XVOLUME to an image file.

Wavelet Toolkit

Widget Commands and Visualization Tools

WV_APPLET - Runs the IDL Wavelet Toolkit GUI.

WV_CW_WAVELET - Compound widget used to select and display wavelet functions.

WV_IMPORT_DATA - Allows user to add a variable to the currently active WV_APPLET widget from the IDL> command prompt.

WV_IMPORT_WAVELET - Allows user to add wavelet functions to the IDL Wavelet Toolkit.

WV_PLOT3D_WPS - Runs the GUI for 3D visualization of the wavelet power spectrum.

WV_PLOT_MULTIRES - Runs GUI for multiresolution analysis.

WV_TOOL_DENOISE - Runs the GUI for wavelet filtering and denoising.

Wavelet Transform

WV_CWT - Returns the one-dimensional continuous wavelet transform of the input array.

WV_DENOISE - Uses the wavelet transform to filter (or de-noise) a multi-dimensional array.

WV_DWT - Returns the multi-dimensional discrete wavelet transform of the input array.

WV_PWT - Returns the partial wavelet transform of the input vector.

Wavelet Functions

WV_FN_COIFLET - Constructs wavelet coefficients for the coiflet wavelet function.

WV_FN_DAUBECHIES - Constructs wavelet coefficients for the Daubechies wavelet function.

WV_FN_GAUSSIAN - Constructs wavelet coefficients for the Gaussian wavelet function.

WV_FN_HAAR - Constructs wavelet coefficients for the Haar wavelet function.

WV_FN_MORLET - Constructs wavelet coefficients for the Morlet wavelet function.

WV_FN_PAUL - Constructs wavelet coefficients for the Paul wavelet function.

WV_FN_SYMLET - Constructs wavelet coefficients for the symlet wavelet function.

Widget Routines

WIDGET_ACTIVEX - Create an ActiveX control and place it into an IDL widget hierarchy.

WIDGET_BASE - Creates base widget (containers for other widgets).

WIDGET_BUTTON - Creates button widgets.

WIDGET_COMBOBOX - Creates editable droplist widgets.

WIDGET_CONTROL - Realizes, manages, and destroys widgets.

WIDGET_DISPLAYCONTEXTMENU - Displays a context-sensitive menu.

WIDGET_DRAW - Creates drawable widgets.

WIDGET_DROPLIST - Creates droplist widgets.

WIDGET_EVENT - Returns events for the widget hierarchy.

WIDGET_INFO - Obtains information about widgets.

WIDGET_LABEL - Creates label widgets.

WIDGET_LIST - Creates list widgets.

WIDGET_PROPERTYSHEET - Creates a property sheet widget, which exposes the properties of an IDL object in a graphical interface. This widget transparently handles property value changes.

WIDGET_SLIDER - Creates slider widgets.

WIDGET_TAB - Creates tab widgets.

WIDGET_TABLE - Creates table widgets.

WIDGET_TEXT - Creates text widgets.

WIDGET_TREE - Creates tree widgets.

XMANAGER - Provides event loop manager for IDL widgets.

XMNG_TMPL - Template for creating widgets.

XMTOOL - Displays tool for viewing XMANAGER widgets.

XREGISTERED - Returns registration status of a given widget.

Widget Routines, Compound

CW_TMPL - Template for compound widgets that use XMANAGER.

Animation

CW_ANIMATE - Creates a compound widget for animation.

CW_ANIMATE_GETP - Gets pixmap window IDs used by CW_ANIMATE.

CW_ANIMATE_LOAD - Loads images into CW_ANIMATE.

CW_ANIMATE_RUN - Displays images loaded into CW_ANIMATE.

Color Manipulation

CW_CLR_INDEX - Creates compound widget for the selection of a color index.

CW_COLORSEL - Creates compound widget that displays all colors in current colormap.

CW_PALETTE_EDITOR_SET - Sets the CW_LIGHT_EDITOR properties.

CW_PALETTE_EDITOR - Creates compound widget to display and edit color palettes.

CW_PALETTE_EDITOR_GET - Gets the CW_PALETTE_EDITOR properties.

CW_PALETTE_EDITOR_SET - Sets the CW_PALETTE_EDITOR properties.

CW_RGBSLIDER - Creates compound widget with sliders for adjusting RGB color values.

Data Entry and Display

CW_FIELD - Creates a widget data entry field.

CW_FILESEL - Creates compound widget for file selection.

CW_FORM - Creates compound widget for creating forms.

Image/Data Manipulation

CW_DEFROI - Creates compound widget used to define region of interest.

CW_LIGHT_EDITOR - Creates compound widget to edit properties of existing IDLgrLight objects in a view.

CW_LIGHT_EDITOR_GET - Gets the CW_LIGHT_EDITOR properties.

CW_ZOOM - Creates widget for displaying zoomed images.

Orientation

CW_ARCBALL - Creates compound widget for intuitively specifying 3D orientations.

CW_ORIENT - Creates compound widget used to interactively adjust the 3D drawing transformation.

User Interface

CW_BGROUP - Creates button group for use as a menu.

CW_FSLIDER - Creates slider that selects floating-point values.

CW_PDMENU - Creates widget pulldown menus.

Window Routines

IDLgrWindow - Represents an on-screen area on a display device that serves as a graphics destination.

WDELETE - Deletes IDL graphics windows.

WINDOW - Creates window for the display of graphics or text.

WSET - Selects the current window.

WSHOW - Exposes or hides the designated window.

Alphabetical List of IDL Routines

This quick reference guide contains an alphabetical listing of all IDL routines. The alphabetical listing contains all functions, procedures, statements, and objects, including the syntax of each.

IDL Syntax Conventions

Function: *Result* = FUNCTION(*Argument1* [, *Argument2*] [, KEYWORD1=*value*] [, /KEYWORD2])

Procedure: PROCEDURE, *Argument1* [, *Argument2*] [, KEYWORD1={*value1* | *value2*}] [, /KEYWORD2]

Statement: IF *expression* THEN *statement* [ELSE *statement*]

Elements of Syntax

Element	Description
[ ] (Square brackets)	Indicates that the contents are optional.
[ ] (Italicized square brackets)	Indicates that the square brackets are part of the statement (used to define an array).
Argument	Arguments are shown in italics, and must be specified in the order listed.
KEYWORD	Keywords are all caps, and can be specified in any order. For functions, all arguments and keywords must be contained within parentheses.
/KEYWORD	Indicates a boolean keyword.
Italics	Indicates arguments, expressions, or statements for which you must provide values.
{ } (Braces)	• Indicates that you must choose one of the values they contain • Encloses a list of possible values, separated by vertical lines (|) • Encloses useful information about a keyword • Defines an IDL structure (this is the only case in which the braces are included in the statement).
| (Vertical lines)	Separates multiple values or keywords.
[, $Value_1$, ... , $Value_n$]	Indicates that any number of values can be specified.
[, $Value_1$, ... , $Value_8$]	Indicates the maximum number of values that can be specified.

Square Brackets ([ ])

- Content between square brackets is optional. Pay close attention to the grouping of square brackets. Consider the following examples:

  ROUTINE_NAME, *Value1* [, *Value2*] [, *Value3*] : You must include *Value1*. You do not have to include *Value2* or *Value3*. *Value2* and *Value3* can be specified independently.

  ROUTINE_NAME, *Value1* [, *Value2*, *Value3*] : You must include *Value1*. You do not have to include *Value2* or *Value3*, but you must include both *Value2* and *Value3*, or neither.

  ROUTINE_NAME [, *Value1* [, *Value2*]] : You can specify *Value1* without specifying *Value2*, but if you specify *Value2*, you must also specify *Value1*.

- Do not include square brackets in your statement unless the brackets are italicized. Consider the following syntax:

  Result = KRIG2D(*Z* [, *X*, *Y*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*])

  An example of a valid statement is:

  R = KRIG2D(Z, X, Y, BOUNDS=[0,0,1,1])
- Note that when [, $Value_1$, ... , $Value_n$] is listed, you can specify any number of arguments. When an explicit number is listed, as in [, $Value_1$, ... , $Value_8$], you can specify only as many arguments as are listed.

Braces ({ })

- For certain keywords, a list of the possible values is provided. This list is enclosed in braces, and the choices are separated by a vertical line (|). Do not include the braces in your statement. For example, consider the following syntax:

  READ_JPEG [, TRUE={1 | 2 | 3}]

  In this example, you must choose either 1, 2, or 3. An example of a valid statement is:

  READ_JPEG, TRUE=1
- Braces are used to enclose the allowable range for a keyword value. Unless otherwise noted, ranges provided are inclusive. Consider the following syntax:

  Result = CVTTOBM(*Array* [, THRESHOLD=*value*{0 to 255}])

  An example of a valid statement is:

  Result = CVTTOBM(A, THRESHOLD=150)
- Braces are also used to provide useful information about a keyword. For example:

  [, LABEL=*n*{label every *n*th gridline}]

  Do not include the braces or their content in your statement.
- Certain keywords are prefaced by X, Y, or Z. Braces are used for these keywords to indicate that you must choose one of the values it contains. For example, [{X | Y}RANGE=*array*] indicates that you can specify either XRANGE=*array* or YRANGE=*array.*
- Note that in IDL, braces are used to define structures. When defining a structure, you *do* want to include the braces in your statement.

Italics

- Italicized words are arguments, expressions, or statements for which you must provide values. The value you provide can be a numerical value, such as 10, an expression, such as DIST(100), or a named variable. For keywords that expect a string value, the syntax is listed as KEYWORD=*string*. The value you provide can be a string, such as 'Hello' (enclosed in single quotation marks), or a variable that holds a string value.
- The italicized values that must be provided for keywords are listed in the most helpful terms possible. For example, [, XSIZE=*pixels*] indicates that the XSIZE keyword expects a value in pixels, while [, ORIENTATION=*ccw_degrees_from_horiz*] indicates that you must provide a value in degrees, measured counter-clockwise from horizontal.

Specifying Keywords

- Certain keywords are boolean, meaning they can be set to either 0 or 1. These keywords are switches used to turn an option on and off. Usually, setting such keywords equal to 1 causes the option to be turned on. Explicitly setting the keyword to 0 (or not including the keyword) turns the option off. All keywords in this reference that are preceded by a slash can be set by prefacing them by the slash. For example, SURFACE, DIST(10), /SKIRT is a shortcut for SURFACE, DIST(10), SKIRT=1. To turn the option back off, you must set the keyword equal to 0, as in SURFACE, DIST(10), SKIRT=0.

  In rare cases, a keyword's default value is 1. In these cases, the syntax is listed as KEYWORD=0, as in SLIDE_IMAGE [, *Image*] [, CONGRID=0]. In this example, CONGRID is set to 1 by default. If you specify CONGRID=0, you can turn it back on by specifying either /CONGRID or CONGRID=1.
- Some keywords are used to obtain values that can be used upon return from the function or procedure. These keywords are listed as KEYWORD=*variable*. Any valid variable name can be used for these keywords, and the variable does not need to be defined first. Note, however that when a keyword calls for a named variable, only a named variable can be used—sending an expression causes an error.

  For example, the WIDGET_CONTROL procedure can return the user values of widgets in a named variable using the GET_UVALUE keyword. To return the user value for a widget ID (contained in the variable `mywidget`) in the variable `userval`, you would use the command:

  ```
  WIDGET_CONTROL, mywidget, GET_UVALUE = userval
  ```

  Upon return from the procedure, userval contains the user value. Note that userval did not have to be defined before the call to WIDGET_CONTROL.
- Some routines have keywords that are mutually exclusive, meaning only one of the keywords can be present in a given statement. These keywords are grouped together, and separated by a vertical line. For example, consider the following syntax:

  PLOT, [*X*,] *Y* [, /DATA | , /DEVICE | , /NORMAL]

  In this example, you can choose either DATA, DEVICE, or NORMAL, but not more than one. An example of a valid statement is:

  ```
  PLOT, SIN(A), /DEVICE
  ```
- Keywords can be abbreviated to their shortest unique length. For example, the XSTYLE keyword can be abbreviated to XST because there are no other keywords in IDL that begin with XST. You cannot shorten XSTYLE to XS, however, because there are other keywords that begin with XS, such as XSIZE.

Alphabetical Listing

The following alphabetical listing contains all IDL functions, procedures, and statements included in IDL version 6.2.

A

A_CORRELATE - Computes autocorrelation.
Result = A_CORRELATE(*X, Lag* [, /COVARIANCE] [, /DOUBLE])

ABS - Returns the absolute value of *X*.
Result = ABS(*X* [, *Thread pool keywords*])

ACOS - Returns the arc-cosine of *X*.
Result = ACOS(*X* [, *Thread pool keywords*])

ADAPT_HIST_EQUAL - Performs adaptive histogram equalization
Result = ADAPT_HIST_EQUAL (*Image* [, CLIP=*value*] [, FCN=*vector*] [, NREGIONS=*nregions*] [, TOP=*value*])

ALOG - Returns the natural logarithm of *X*.
Result = ALOG(*X* [, *Thread pool keywords*])

ALOG10 - Returns the logarithm to the base 10 of *X*.
Result = ALOG10(*X* [, *Thread pool keywords*])

AMOEBA - Minimizes a function using downhill simplex method.
Result = AMOEBA(*Ftol* [, FUNCTION_NAME=*string*] [, FUNCTION_VALUE=*variable*] [, NCALLS=*value*] [, NMAX=*value*] [, P0=*vector*, SCALE=*vector* | , SIMPLEX=*array*])

ANNOTATE - Starts IDL widget used to interactively annotate images and plots with text and drawings.
ANNOTATE [, COLOR_INDICES=*array*] [, DRAWABLE=*widget_id* | , WINDOW=*index*] [, LOAD_FILE=*filename*] [, /TEK_COLORS]

APP_USER_DIR - Provides access to the *application user directory.*
Result = APP_USER_DIR(*AuthorDirname*, *AuthorDesc*, *AppDirname*, *AppDesc*, *AppReadmeText*, *AppReadmeVersion* [, AUTHOR_README_TEXT=*string array*] [, AUTHOR_README_VERSION=string] [, RESTRICT_APPVERSION=string] [, /RESTRICT_ARCH] [, /RESTRICT_FAMILY | /RESTRICT_OS] [, /RESTRICT_FILE_OFFSET_BITS] [, /RESTRICT_IDL_RELEASE] [, /RESTRICT_MEMORY_BITS])

APP_USER_DIR_QUERY - Allows searches for *application user directories.*
Result = APP_USER_DIR_QUERY(*AuthorDirname*, *AppDirname* [, COUNT=variable] [, /EXCLUDE_CURRENT] [RESTRICT keywords] [QUERY keywords])

ARG_PRESENT - Returns TRUE if the value of the specified variable can be passed back to the caller.
Result = ARG_PRESENT(*Variable*)

ARRAY_EQUAL - Provides a fast way to compare data for equality in situations where the index of the elements that differ are not of interest.
Result = ARRAY_EQUAL(*Op1* , *Op2* [, /NO_TYPECONV])

ARRAY_INDICES - Converts one-dimensional subscripts of an array into corresponding multi-dimensional subscripts.
Result = ARRAY_INDICES(*Array, Index* [, /DIMENSIONS])

ARROW - Draws line with an arrow head.
ARROW, *X0*, *Y0*, *X1*, *Y1* [, /DATA | , /NORMALIZED] [, HSIZE=*length*] [, COLOR=*index*] [, HTHICK=*value*] [, /SOLID] [, THICK=*value*]

ASCII_TEMPLATE - Presents a GUI that generates a template defining an ASCII file format.
Result = ASCII_TEMPLATE([*Filename*] [, BROWSE_LINES=*lines*] [, CANCEL=*variable*] [, GROUP=*widget_id*])

ASIN - Returns the arc-sine of *X*.
Result = ASIN(*X* [, *Thread pool keywords*])

ASSOC - Associates an array structure with a file.
Result = ASSOC(*Unit*, *Array_Structure* [, *Offset*] [, /PACKED])

ATAN - Returns the arc-tangent of *X*.
Result = ATAN(*X* [, /PHASE] [, *Thread pool keywords*])
or
Result = ATAN(*Y, X*) [, *Thread pool keywords*])

AXIS - Draws an axis of the specified type and scale.

AXIS [, *X* [, *Y* [, *Z*]]] [, /SAVE] [, XAXIS={0 | 1} | YAXIS={0 | 1} | ZAXIS={0 | 1 | 2 | 3}] [, /XLOG] [, /YNOZERO] [, /YLOG] [, /ZLOG]

Graphics Keywords: [, CHARSIZE=*value*] [, CHARTHICK=*integer*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, /NODATA] [, /NOERASE] [, SUBTITLE=*string*] [, /T3D] [, TICKLEN=*value*] [, {X | Y | Z}CHARSIZE=*value*] [, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}] [, {X | Y | Z}MARGIN=*[left, right]*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=*[min, max]*] [, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*] [, {X | Y | Z}TICKFORMAT=*string or a vector of strings*] [, {X | Y | Z}TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT=*scalar*] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string_array*] [, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKUNITS=*string or a vector of strings*] [, {X | Y | Z}TICKV=*array*] [, {X | Y | Z}TICK_GET=*variable*] [, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

B

BAR_PLOT - Creates a bar graph.

BAR_PLOT, *Values* [, BACKGROUND=*color_index*] [, BARNAMES=*string_array*] [, BAROFFSET=*scalar*] [, BARSPACE=*scalar*] [, BARWIDTH=*value*] [, BASELINES=*vector*] [, BASERANGE=*scalar*{0.0 to 1.0}] [, COLORS=*vector*] [, /OUTLINE] [, /OVERPLOT] [, /ROTATE] [, TITLE=*string*] [, XTITLE=*string*] [, YTITLE=*string*]

BEGIN...END - Defines a block of statements.

BEGIN
statements
END | ENDIF | ENDELSE | ENDFOR | ENDREP | ENDWHILE

BESELI - Returns the I Bessel function of order *N* for *X*.

Result = BESELI(*X, N* [, /DOUBLE] [, ITER=*variable*])

BESELJ - Returns the J Bessel function of order *N* for *X*.

Result = BESELJ(*X, N* [, /DOUBLE] [, ITER=*variable*])

BESELK - Returns the K Bessel function of order *N* for the *X*.

Result = BESELK(*X, N* [, /DOUBLE] [, ITER=*variable*])

BESELY - Returns the Y Bessel function of order *N* for *X*.

Result = BESELY(*X, N* [, /DOUBLE] [, ITER=*variable*])

BETA - Returns the value of the beta function.

Result = BETA(*Z, W* [, /DOUBLE])

BILINEAR - Computes array using bilinear interpolation.

Result = BILINEAR(*P, IX, JY* [, MISSING=*value*])

BIN_DATE - Converts ASCII date/time string to binary string.

Result = BIN_DATE(*Ascii_Time*)

BINARY_TEMPLATE - Presents a GUI for interactively generating a template structure for use with READ_BINARY.

Template = BINARY_TEMPLATE ([*Filename*] [, CANCEL=*variable*] [, GROUP=*widget_id*] [, N_ROWS=*rows*] [, TEMPLATE=*variable*])

BINDGEN - Returns byte array with each element set to its subscript.

Result = BINDGEN(D_1 *[, ...,*D_8*]* [, *Thread pool keywords*])

BINOMIAL - Computes binomial distribution function.

Result = BINOMIAL(*V, N, P* [, /DOUBLE] [, /GAUSSIAN])

BIT_FFS - Returns the index of the first bit set (non-zero) in an integer.

Result = BIT_FFS(*Value*)

BIT_POPULATION - Returns the number of set (non-zero) bits in an integer.

Result = BIT_POPULATION(*Value*)

BLAS_AXPY - Updates existing array by adding a multiple of another array.

BLAS_AXPY, *Y, A, X* [, *D1, Loc1* [, *D2, Range*]]

BLK_CON - Convolves input signal with impulse-response sequence.

Result = BLK_CON(*Filter, Signal* [, B_LENGTH=*scalar*] [, /DOUBLE])

BOX_CURSOR - Emulates the operation of a variable-sized box cursor.

BOX_CURSOR, [*X0, Y0, NX, NY* [, /INIT] [, /FIXED_SIZE]] [, /MESSAGE]

BREAK - Immediately exits from a loop (FOR, WHILE, REPEAT), CASE, or SWITCH statement.

BREAK

BREAKPOINT - Sets and clears breakpoints for debugging.

BREAKPOINT [, *File*], *Index* [, AFTER=*integer*] [, /CLEAR] [, CONDITION='*expression*'] [, /DISABLE] [, /ENABLE] [, /ON_RECOMPILE] [, /ONCE] [, /SET]

BROYDEN - Solves nonlinear equations using Broyden's method.

Result = BROYDEN(*X, Vecfunc* [, CHECK=*variable*] [, /DOUBLE] [, EPS=*value*] [, ITMAX=*value*] [, STEPMAX=*value*] [, TOLF=*value*] [, TOLMIN=*value*] [, TOLX=*value*])

BYTARR - Creates a byte vector or array.

Result = BYTARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

BYTE - Converts argument to byte type.
Result = BYTE(*Expression [, Offset [, D_1 [, ...,D_8]]]* [, *Thread pool keywords*])

BYTEORDER - Converts between host and network byte ordering.
BYTEORDER, *Variable$_1$, ..., Variable$_n$* [, /DTOVAX] [, /DTOXDR] [, /FTOVAX] [, /FTOXDR] [, /HTONL] [, /HTONS] [, /L64SWAP] [, /LSWAP] [, /NTOHL] [, /NTOHS] [, /SSWAP] [, /SWAP_IF_BIG_ENDIAN] [, /SWAP_IF_LITTLE_ENDIAN] [, /VAXTOD] [, /VAXTOF] [, /XDRTOD] [, /XDRTOF] [, *Thread pool keywords*]

BYTSCL - Scales all values of an array into range of bytes.
Result = BYTSCL(*Array* [, MAX=*value*] [, MIN=*value*] [, /NAN] [, TOP=*value*] [, *Thread pool keywords*])

C

C_CORRELATE - Computes cross correlation.
Result = C_CORRELATE(*X, Y, Lag* [, /COVARIANCE] [, /DOUBLE])

CALDAT - Converts Julian date to month, day, year.
CALDAT, *Julian, Month* [, *Day* [, *Year* [, *Hour* [, *Minute* [, *Second*]]]]]

CALENDAR - Displays a calendar for a given month or year.
CALENDAR [[, *Month*] , *Year*]

CALL_EXTERNAL - Calls a function in an external sharable object and returns a scalar value.
Result = CALL_EXTERNAL(*Image*, *Entry* [, P_0, ..., P_{N-1}] [, /ALL_VALUE] [, /B_VALUE | , /D_VALUE | , /F_VALUE | , /I_VALUE | , /L64_VALUE | , /S_VALUE | , /UI_VALUE | , /UL_VALUE | , /UL64_VALUE] [, /CDECL] [, RETURN_TYPE=*value*] [, /UNLOAD] [, VALUE=*byte_array*] [, WRITE_WRAPPER=*wrapper_file*])
Auto Glue keywords: [, /AUTO_GLUE] [, CC=*string*] [, COMPILE_DIRECTORY=*string*] [, EXTRA_CFLAGS=*string*] [, EXTRA_LFLAGS=*string*] [, /IGNORE_EXISTING_GLUE] [, LD=*string*] [, /NOCLEANUP] [, /SHOW_ALL_OUTPUT] [, /VERBOSE]

CALL_FUNCTION - Calls an IDL function.
Result = CALL_FUNCTION(*Name* [, P_1, ..., P_n])

CALL_METHOD - Calls an IDL object method.
CALL_METHOD, *Name, ObjRef,* [, P_1, ..., P_n] or
Result = CALL_METHOD(*Name, ObjRef,* [, P_1, ..., P_n])

CALL_PROCEDURE - Calls an IDL procedure.
CALL_PROCEDURE, *Name* [, P_1, ..., P_n]

CASE - Selects one statement for execution, depending on the value of an expression.
CASE *expression* OF
expression: *statement*
...
expression: *statement*
[ELSE: *statement*]
ENDCASE

CATCH - Declares and clears exception handlers.
CATCH, [*Variable*] [, /CANCEL]

CD - Sets and/or changes the current working directory.
CD [, *Directory*] [, CURRENT=*variable*]

CDF_* Routines - See "CDF Routines" on page 126.

CEIL - Returns the closest integer greater than or equal to *X*.
Result = CEIL(*X* [, /L64] [, *Thread pool keywords*])

CHEBYSHEV - Returns the forward or reverse Chebyshev polynomial expansion.
Result = CHEBYSHEV(*D, N*)

CHECK_MATH - Returns and clears accumulated math error status.
Result = CHECK_MATH([, MASK=*bitmask*] [, /NOCLEAR] [, /PRINT])

CHISQR_CVF - Computes cutoff value in a Chi-square distribution.
Result = CHISQR_CVF(*P, Df*)

CHISQR_PDF - Computes Chi-square distribution function.
Result = CHISQR_PDF(*V, Df*)

CHOLDC - Constructs Cholesky decomposition of a matrix.
CHOLDC, *A, P* [, /DOUBLE]

CHOLSOL - Solves set of linear equations (use with CHOLDC).
Result = CHOLSOL(*A, P, B* [, /DOUBLE])

CINDGEN - Returns a complex array with each element set to its subscript.
Result = CINDGEN(*D_1 [, ...,D_8]* [, *Thread pool keywords*])

CIR_3PNT - Returns radius and center of circle, given 3 points.
CIR_3PNT, *X, Y, R, X0, Y0*

CLOSE - Closes the specified files.
CLOSE[, *Unit$_1$, ..., Unit$_n$*] [, /ALL] [, EXIT_STATUS=*variable*] [, /FILE] [, /FORCE]

CLUST_WTS - Computes the cluster weights of an array for cluster analysis.
Result = CLUST_WTS(*Array* [, /DOUBLE] [, N_CLUSTERS=*value*] [, N_ITERATIONS=*integer*] [, VARIABLE_WTS=*vector*])

CLUSTER - Performs cluster analysis.
Result = CLUSTER(*Array, Weights* [, /DOUBLE] [, N_CLUSTERS=*value*])

CLUSTER_TREE - Computes the hierarchical clustering for a set of *m* items in an *n*-dimensional space.

Result = CLUSTER_TREE(*Pairdistance*, *Linkdistance* [, LINKAGE = *value*])

or for LINKAGE = 3 (centroid):

Result = CLUSTER_TREE(*Pairdistance*, *Linkdistance*, LINKAGE = 3, DATA = *array*[, MEASURE=*value*] [, POWER_MEASURE=*value*])

CMYK_CONVERT - Converts color triples to and from RGB and CMYK.

CMYK_CONVERT, C, M, Y, K, R, G, B, [, /TO_CMYK]

COLOR_CONVERT - Converts color triples to and from RGB, HLS, and HSV.

COLOR_CONVERT, I_0, I_1, I_2, O_0, O_1, O_2 {, /HLS_RGB | , /HSV_RGB | , /RGB_HLS | , /RGB_HSV}

COLOR_QUAN - Converts true-color (24-bit) image to pseudo-color (8-bit) image.

Result = COLOR_QUAN(*Image_R*, *Image_G*, *Image_B*, *R*, *G*, *B*)

or

Result = COLOR_QUAN(*Image*, *Dim*, *R*, *G*, *B*)

Keywords: [, COLORS=*integer*{2 to 256}] [, CUBE={2 | 3 | 4 | 5 | 6} | , GET_TRANSLATION=*variable*] [, /MAP_ALL]] [, /DITHER] [, ERROR=*variable*] [, TRANSLATION=*vector*]

COLORMAP_APPLICABLE - Determines whether the current visual class supports the use of a colormap.

Result = COLORMAP_APPLICABLE(*redrawRequired*)

COMFIT - Fits paired data using one of six common filtering functions.

Result = COMFIT(*X*, *Y*, *A* {, /EXPONENTIAL | , /GEOMETRIC | , /GOMPERTZ | , /HYPERBOLIC | , /LOGISTIC | , /LOGSQUARE} [, SIGMA=*variable*] [, WEIGHTS=*vector*] [, YFIT=*variable*])

COMMAND_LINE_ARGS - returns string values supplied when the user starts IDL with the -arg or -args command line options.

Result = COMMAND_LINE_ARGS([COUNT=*variable*])

COMMON - Creates a common block.

COMMON *Block_Name*, *Variable*$_1$, ..., *Variable*$_n$

COMPILE_OPT - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

COMPILE_OPT *opt*$_1$ [, *opt*$_2$, ..., *opt*$_n$]

Note: *opt*$_n$ can be IDL2, DEFINT32, HIDDEN, LOGICAL_PREDICATE, OBSOLETE, STRICTARR, or STRICTARRSUBS

COMPLEX - Converts argument to complex type.

Result = COMPLEX(*Real* [, *Imaginary*] [, /DOUBLE] [, *Thread pool keywords*])

or

Result = COMPLEX(*Expression*, *Offset*, D_1 *[, ...,*D_8*]* [, ./DOUBLE] [, *Thread pool keywords*])

COMPLEXARR - Creates a complex, single-precision, floating-point vector or array.

Result = COMPLEXARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

COMPLEXROUND - Rounds a complex array.

Result = COMPLEXROUND(*Input*)

COMPUTE_MESH_NORMALS - Computes normal vectors for a set of polygons.

Result=COMPUTE_MESH_NORMALS(*fVerts*[, *iConn*])

COND - Computes the condition number of a square matrix.

Result = COND(*A* [, /DOUBLE] [, LNORM={0 | 1 | 2}])

CONGRID - Resamples an image to any dimensions.

Result = CONGRID(*Array, X, Y, Z* [, /CENTER] [, CUBIC=*value*{-1 to 0}] [, /INTERP] [, /MINUS_ONE])

CONJ - Returns the complex conjugate of *X*.

Result = CONJ(*X* [, *Thread pool keywords*])

CONSTRAINED_MIN - Minimizes a function using Generalized Reduced Gradient Method.

CONSTRAINED_MIN, *X*, *Xbnd*, *Gbnd*, *Nobj*, *Gcomp*, *Inform* [, ESPTOP=*value*] [, LIMSER=*value*] [, /MAXIMIZE] [, NSTOP=*value*] [, REPORT=*filename*] [, TITLE=*string*]

CONTINUE - Immediately starts the next iteration of the enclosing FOR, WHILE, or REPEAT loop.

CONTINUE

CONTOUR - Draws a contour plot.

CONTOUR, *Z* [, *X*, *Y*] [, C_CHARSIZE=*value*] [, C_CHARTHICK=*integer*] [, C_COLORS=*vector*] [, C_LABELS=*vector*{each element 0 or 1}] [, C_LINESTYLE=*vector*] [{, /FILL | , /CELL_FILL} | [, C_ANNOTATION=*vector_of_strings*] [, C_ORIENTATION=*degrees*] [, C_SPACING=*value*]] [, C_THICK=*vector*] [, /CLOSED] [, /DOWNHILL] [, /FOLLOW] [, /IRREGULAR] [, /ISOTROPIC] [, LEVELS=*vector* | NLEVELS=*integer*{1 to 60}] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, /OVERPLOT] [{, /PATH_DATA_COORDS, PATH_FILENAME=*string*, PATH_INFO=*variable*, PATH_XY=*variable*} | , TRIANGULATION=*variable*] [, /PATH_DOUBLE] [, /XLOG] [, /YLOG] [, ZAXIS={0 | 1 | 2 | 3 | 4}] [, /ZLOG]

Graphics Keywords: Accepts all PLOT graphics keywords except for: LINESTYLE, PSYM, SYMSIZE.

CONVERT_COORD - Transforms coordinates to and from the coordinate systems supported by IDL.

Result = CONVERT_COORD(*X* [, *Y* [, *Z*]] [, /DATA | , /DEVICE | , /NORMAL] [, /DOUBLE] [, /T3D] [, /TO_DATA | , /TO_DEVICE | , /TO_NORMAL])

CONVOL - Convolves two vectors or arrays.
Result = CONVOL(*Array, Kernel* [, *Scale_Factor*] [, BIAS=*value*] [, /CENTER] [, /EDGE_WRAP] [, /EDGE_TRUNCATE] [, /EDGE_ZERO] [, INVALID=*value*] [, MISSING=*value*] [, /NAN] [, /NORMALIZE] [, *Thread pool keywords*])

COORD2TO3 - Returns 3D data coordinates given normalized screen coordinates.
Result = COORD2TO3(*Mx, My, Dim, D0* [, *PTI*])

COPY_LUN - Copies data between two open files.
COPY_LUN, *FromUnit, ToUnit* [, *Num*] [, /EOF] [, /LINES] [, TRANSFER_COUNT=*va;ie*]

CORRELATE - Computes the linear Pearson correlation.
Result = CORRELATE(*X* [, *Y*] [, /COVARIANCE] [, /DOUBLE])

COS - Returns the cosine of X.
Result = COS(*X* [, *Thread pool keywords*])

COSH - Returns the hyperbolic cosine of *X*.
Result = COSH(*X* [, *Thread pool keywords*])

CPU - Changes the values stored in the read-only !CPU system variable.
CPU [, /RESET] [, RESTORE=*structure*] [,TPOOL_MAX_ELTS = *NumMaxElts*] [,TPOOL_MIN_ELTS = *NumMinElts*] [,TPOOL_NTHREADS = *NumThreads*] [,/VECTOR_ENABLE]

CRAMER - Solves system of linear equations using Cramer's rule.
Result = CRAMER(*A, B* [, /DOUBLE] [, ZERO=*value*])

CREATE_CURSOR - Returns a 16x16 image suitable for a cursor from the input string array.
Result = CREATE_CURSOR(*StringArray* [, HOTSPOT=*variable*] [, MASK=*variable*])

CREATE_STRUCT - Creates and concatenates structures.
Result = CREATE_STRUCT([Tag_1, $Value_1$, ..., Tag_n, $Value_n$] [, NAME=*string*])
or
Result = CREATE_STRUCT(*[*Tag_1, ..., Tag_n*]*, $Value_1$, ..., $Value_n$ [, NAME=*string*])

CREATE_VIEW - Sets up 3D transformations.
CREATE_VIEW [, AX=*value*] [, AY=*value*] [, AZ=*value*] [, PERSP=*value*] [, /RADIANS] [, WINX=*pixels*] [, WINY=*pixels*] [, XMAX=*scalar*] [, XMIN=*scalar*] [, YMAX=*scalar*] [, YMIN=*scalar*] [, ZFAC=*value*] [, ZMAX=*scalar*] [, ZMIN=*scalar*] [, ZOOM=*scalar or 3-element vector*]

CROSSP - Computes vector cross product.
Result = CROSSP(*V1, V2*)

CRVLENGTH - Computes the length of a curve.
Result = CRVLENGTH(*X, Y* [, /DOUBLE])

CT_LUMINANCE - Calculates the luminance of colors.
Result = CT_LUMINANCE([*R, G, B*] [, BRIGHT=*variable*] [, DARK=*variable*] [, /READ_TABLES])

CTI_TEST - Performs chi-square goodness-of-fit test.
Result = CTI_TEST(*Obfreq* [, COEFF=*variable*] [, /CORRECTED] [, CRAMV=*variable*] [, DF=*variable*] [, EXFREQ=*variable*] [, RESIDUAL=*variable*])

CURSOR - Reads position of the interactive graphics cursor.
CURSOR, *X, Y* [, *Wait* | [, /CHANGE | , /DOWN | , /NOWAIT | , /UP | , /WAIT]] [, /DATA | , /DEVICE, | , /NORMAL]

CURVEFIT - Fits multivariate data with a user-supplied function.
Result = CURVEFIT(*X, Y, Weights, A* [, *Sigma*] [, CHISQ=*variable*] [, /DOUBLE] [, FITA=*vector*] [, FUNCTION_NAME=*string*] [, ITER=*variable*] [, ITMAX=*value*] [, /NODERIVATIVE] [, STATUS={0 | 1 | 2}] [, TOL=*value*] [, YERROR=*variable*])

CV_COORD - Converts 2D and 3D coordinates between coordinate systems.
Result = CV_COORD([, /DEGREES] [, /DOUBLE] [, FROM_CYLIN=*cyl_coords* | , FROM_POLAR=*pol_coords* | , FROM_RECT=*rect_coords* | , FROM_SPHERE=*sph_coords*] [, /TO_CYLIN | , /TO_POLAR | , /TO_RECT | , /TO_SPHERE])

CVTTOBM - Creates a bitmap byte array for a button label.
Result = CVTTOBM(*Array* [, THRESHOLD=*value*{0 to 255}])

CW_ANIMATE - Creates a compound widget for animation.
Result = CW_ANIMATE(*Parent, Sizex, Sizey, Nframes* [, /NO_KILL] [, OPEN_FUNC=*string*] [, PIXMAPS=*vector*] [, TAB_MODE=*value*] [, /TRACK] [, UNAME=*string*] [, UVALUE=*value*])

CW_ANIMATE_GETP - Gets pixmap window IDs used by CW_ANIMATE.
CW_ANIMATE_GETP, *Widget, Pixmaps* [, /KILL_ANYWAY]

CW_ANIMATE_LOAD - Loads images into CW_ANIMATE.
CW_ANIMATE_LOAD, *Widget* [, /CYCLE] [, FRAME=*value*{0 to *NFRAMES*}] [, IMAGE=*value*] [, /ORDER] [, WINDOW=*[window_num* [, *X0, Y0, Sx, Sy]]*] [, XOFFSET=*pixels*] [, YOFFSET=*pixels*]

CW_ANIMATE_RUN - Displays images loaded into CW_ANIMATE.
CW_ANIMATE_RUN, *Widget* [, *Rate*{0 to 100}] [, NFRAMES=*value*] [, /STOP]

CW_ARCBALL - Creates compound widget for intuitively specifying 3D orientations.

Result = CW_ARCBALL(*Parent* [, COLORS=*array*] [, /FRAME] [, LABEL=*string*] [, RETAIN={0 | 1 | 2}] [, SIZE=*pixels*] [, TAB_MODE=*value*] [, /UPDATE] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*array*])

CW_BGROUP - Creates button group for use as a menu.

Result = CW_BGROUP(*Parent*, *Names* [, BUTTON_UVALUE=*array*] [, COLUMN=*value*] [, EVENT_FUNC=*string*] [{, /EXCLUSIVE | , /NONEXCLUSIVE} | [, SPACE=*pixels*] [, XPAD=*pixels*] [, YPAD=*pixels*]] [, FONT=*font*] [, FRAME=*width*] [, IDS=*variable*] [, LABEL_LEFT=*string* | , LABEL_TOP=*string*] [, /MAP] [, /NO_RELEASE] [, /RETURN_ID | , /RETURN_INDEX | , /RETURN_NAME] [, ROW=*value*] [, /SCROLL] [, SET_VALUE=*value*] [, SPACE=*value*] [, TAB_MODE=*value*] [, X_SCROLL_SIZE=*width*] [, Y_SCROLL_SIZE=*height*] [, SET_VALUE=*value*] [, UNAME=*string*] [, UVALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*width*] [, YOFFSET=*value*] [, YSIZE=*value*])

CW_CLR_INDEX - Creates compound widget to select color index.

Result = CW_CLR_INDEX(*Parent* [, COLOR_VALUES=*vector* | [, NCOLORS=*value*] [, START_COLOR=*value*]] [, EVENT_FUNC='*function_name*'] [, /FRAME] [, LABEL=*string*] [, NCOLORS=*value*] [, START_COLOR=*value*] [, TAB_MODE=*value*] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*value*] [, XSIZE=*pixels*] [, YSIZE=*pixels*])

CW_COLORSEL - Creates compound widget that displays all colors in current colormap.

Result = CW_COLORSEL(*Parent* [, /FRAME] [, TAB_MODE=*value*] [, UNAME=*string*] [, UVALUE=*value*] [, XOFFSET=*value*] [, YOFFSET=*value*])

CW_DEFROI - Creates compound widget used to define region of interest.

Result = CW_DEFROI(*Draw* [, IMAGE_SIZE=*vector*] [, OFFSET=*vector*] [, /ORDER] [, /RESTORE] [, TAB_MODE=*value*] [, ZOOM=*vector*])

CW_FIELD - Creates a widget data entry field.

Result = CW_FIELD(*Parent* [, /ALL_EVENTS] [, /COLUMN] [, FIELDFONT=*font*] [, /FLOATING | , /INTEGER | , /LONG | , /STRING] [, FONT=*string*] [, FRAME=*pixels*] [, /NOEDIT] [, /RETURN_EVENTS] [, /ROW] [, STRING=*string*] [, TAB_MODE=*value*] [, /TEXT_FRAME] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*value*] [, XSIZE=*characters*] [, YSIZE=*lines*])

CW_FILESEL - Creates compound widget for file selection.

Result = CW_FILESEL (*Parent* [, /FILENAME] [, FILTER=*string array*] [, /FIX_FILTER] [, /FRAME] [, /IMAGE_FILTER] [, /MULTIPLE | , /SAVE] [, PATH=*string*] [, TAB_MODE=*value*] [, UNAME=*string*] [, UVALUE=*value*] [, /WARN_EXIST])

CW_FORM - Creates compound widget for creating forms.

Result = CW_FORM([*Parent*,] *Desc* [, /COLUMN] [, IDS=*variable*] [, TAB_MODE=*value*] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*])

Note: *Desc* is a string array. Each element of string array contains 2 or more comma-delimited fields. Each string has the following format: *['Depth, Item, Initial_Value, Keywords']*

CW_FSLIDER - Creates slider that selects floating-point values.

Result = CW_FSLIDER(*Parent* [, /DOUBLE] [, /DRAG] [, /EDIT] [, FORMAT=*string*] [, /FRAME] [, MAXIMUM=*value*] [, MINIMUM=*value*] [, SCROLL=*units*] [, /SUPRESS_VALUE] [, TAB_MODE=*value*] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*initial_value*] [, XSIZE=*length* | {, /VERTICAL [, YSIZE=*height*]}])

CW_LIGHT_EDITOR - Creates compound widget to edit properties of existing IDLgrLight objects in a view.

Result = CW_LIGHT_EDITOR (*Parent* [, /DIRECTION_DISABLED] [, /DRAG_EVENTS] [, FRAME=*width*] [, /HIDE_DISABLED] [, LIGHT=*objref(s)*] [, /LOCATION_DISABLED] [, TAB_MODE=*value*] [, /TYPE_DISABLED] [, UVALUE=*value*] [, XSIZE=*pixels*] [, YSIZE=*pixels*] [, XRANGE=*vector*] [, YRANGE=*vector*] [, ZRANGE=*vector*])

CW_LIGHT_EDITOR_GET - Gets the CW_LIGHT_EDITOR properties.

CW_ LIGHT_EDITOR_GET, *WidgetID* [, DIRECTION_DISABLED=*variable*] [, DRAG_EVENTS=*variable*] [, HIDE_DISABLED=*variable*] [, LIGHT=*variable*] [, LOCATION_DISABLED=*variable*] [, TYPE_DISABLED=*variable*] [, XSIZE=*variable*] [, YSIZE=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

CW_LIGHT_EDITOR_SET - Sets the CW_LIGHT_EDITOR properties.

CW_ LIGHT_EDITOR_SET, *WidgetID*
[, /DIRECTION_DISABLED] [, /DRAG_EVENTS]
[, /HIDE_DISABLED] [, LIGHT=*objref(s)*]
[, /LOCATION_DISABLED] [, /TYPE_DISABLED]
[, XSIZE=*pixels*] [, YSIZE=*pixels*] [, XRANGE=*vector*]
[, YRANGE=*vector*] [, ZRANGE=*vector*]

CW_ORIENT - Creates compound widget used to interactively adjust the 3D drawing transformation.

Result = CW_ORIENT(*Parent* [, AX=*degrees*]
[, AZ=*degrees*] [, /FRAME] [, TAB_MODE=*value*]
[, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*]
[, XSIZE=*width*] [, YSIZE=*height*])

CW_PALETTE_EDITOR - Creates compound widget to display and edit color palettes.

Result = CW_PALETTE_EDITOR (*Parent*
[, DATA=*array*] [, FRAME=*width*]
[, HISTOGRAM=*vector*] [, /HORIZONTAL]
[, SELECTION=*[start, end]*] [, TAB_MODE=*value*]
[, UNAME=*string*] [, UVALUE=*value*] [, XSIZE=*width*]
[, YSIZE=*height*])

CW_PALETTE_EDITOR_GET - Gets the CW_PALETTE_EDITOR properties.

CW_PALETTE_EDITOR_GET, *WidgetID*
[, ALPHA=*variable*] [, HISTOGRAM=*variable*]

CW_PALETTE_EDITOR_SET - Sets the CW_PALETTE_EDITOR properties.

CW_PALETTE_EDITOR_SET, *WidgetID*
[, ALPHA=*byte_vector*] [, HISTOGRAM=*byte_vector*]

CW_PDMENU - Creates widget pulldown menus.

Result = CW_PDMENU(*Parent*, *Desc* [, /COLUMN]
[, /CONTEXT_MENU] [, DELIMITER=*string*]
[, FONT=*value*] [, /MBAR [, /HELP]] [, IDS=*variable*]
[, /RETURN_ID | , /RETURN_INDEX | ,
/RETURN_NAME | , /RETURN_FULL_NAME]
[, TAB_MODE=*value*] [, UNAME=*string*]
[, UVALUE=*value*] [, XOFFSET=*value*]
[, YOFFSET=*value*])

CW_RGBSLIDER - Creates compound widget with sliders for adjusting RGB color values.

Result = CW_RGBSLIDER(*Parent*
[, /CMY | , /HSV | , /HLS | , /RGB]
[, /COLOR_INDEX | , GRAPHICS_LEVEL={1 | 2}]
[, /DRAG] [, /FRAME] [, LENGTH=*value*] [, /RGB]
[, TAB_MODE=*value*] [, UNAME=*string*]
[, UVALUE=*value*] [, VALUE=*[r, g, b]*]
[, /VERTICAL])

CW_TMPL - Template for compound widgets that use XMANAGER.

Result = CW_TMPL(*Parent* [, TAB_MODE=*value*]
[, UNAME=*string*] [, UVALUE=*value*])

CW_ZOOM - Creates widget for displaying zoomed images.

Result = CW_ZOOM(*Parent* [, /FRAME] [, MAX=*scale*]
[, MIN=*scale*] [, RETAIN={0 | 1 | 2}] [, SAMPLE=*value*]
[, SCALE=*value*] [, TAB_MODE=*value*] [, /TRACK]
[, UNAME=*string*] [, UVALUE=*value*] [, XSIZE=*width*]
[, X_SCROLL_SIZE=*width*] [, X_ZSIZE=*zoom_width*]
[, YSIZE=*height*] [, Y_SCROLL_SIZE=*height*]
[, Y_ZSIZE=*zoom_height*])

D

DBLARR - Creates a double-precision array.

Result = DBLARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

DCINDGEN - Returns a double-precision, complex array with each element set to its subscript.

Result = DCINDGEN(D_1 *[, ...,*D_8*]* [, *Thread pool keywords*])

DCOMPLEX - Converts argument to double-precision complex type.

Result = DCOMPLEX(*Real* [, *Imaginary*] [, *Thread pool keywords*])
or
Result = DCOMPLEX(*Expression*, *Offset*, D_1 *[, ...,*D_8*]*
[, *Thread pool keywords*])

DCOMPLEXARR - Creates a complex, double-precision vector or array.

Result = DCOMPLEXARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

DEFINE_KEY - Programs keyboard function keys.

DEFINE_KEY, *Key* [, *Value*] [, /MATCH_PREVIOUS]
[, /NOECHO] [, /TERMINATE]
UNIX Keywords: [, /BACK_CHARACTER]
[, /BACK_WORD] [, /CONTROL | , /ESCAPE]
[, /DELETE_CHARACTER] [, /DELETE_CURRENT]
[, /DELETE_EOL] [, /DELETE_LINE]
[, /DELETE_WORD] [, /END_OF_LINE]
[, /END_OF_FILE] [, /ENTER_LINE]
[, /FORWARD_CHARACTER]
[, /FORWARD_WORD]
[, /INSERT_OVERSTRIKE_TOGGLE]
[, /NEXT_LINE] [, /PREVIOUS_LINE] [, /RECALL]
[, /REDRAW] [, /START_OF_LINE]

DEFINE_MSGBLK - Defines and loads a new message block into the current IDL session.

DEFINE_MSGBLK, *BlockName*, *ErrorNames*,
ErrorFormats [, /IGNORE_DUPLICATE]
[, PREFIX = *PrefixStr*]

DEFINE_MSGBLK_FROM_FILE - Reads the definition of a message block from a file, and loads it into the current IDL session.

DEFINE_MSGBLK_FROM_FILE, *Filename*
[, BLOCK = *BlockName*] [, /IGNORE_DUPLICATE]
[, PREFIX = *PrefixStr*] [, /VERBOSE]

DEFROI - Defines an irregular region of interest of an image.

Result = DEFROI(*Sx, Sy* [, *Xverts*, *Yverts*] [, /NOREGION] [, /NOFILL] [, /RESTORE] [, X0=*device_coord*, Y0=*device_coord*] [, ZOOM=*factor*])

DEFSYSV - Creates a new system variable.

DEFSYSV,*Name*,*Value* [, *Read_Only*] [, EXISTS=*variable*]

DELVAR - Deletes variables from the main IDL program level.

DELVAR, V_1, ..., V_n

DENDRO_PLOT - Draws a two-dimensional dendrite plot on the current direct graphics device if given a hierarchical tree cluster, as created by CLUSTER_TREE.

DENDRO_PLOT, *Clusters*, *Linkdistance* [, LABEL_CHARSIZE=*value*] [, LABEL_CHARTHICK=*value*] [, LABEL_COLOR=*value*] [, LABEL_NAMES=*vector*] [, LABEL_ORIENTATION=*value*] [, LINECOLOR=*value*] [, LINESTYLE=*value*] [, ORIENTATION={1|2|3|4}] [, /OVERPLOT]

DENDROGRAM - Constructs a dendrogram and returns a set of vertices and connectivity that can be used to visualize the dendrite plot if given a hierarchical tree cluster, as created by CLUSTER_TREE.

DENDROGRAM, *Clusters*, *Linkdistance*, *Outverts*, *Outconn* [, LEAFNODES=*variable*]

DERIV - Performs differentiation using 3-point, Lagrangian interpolation and returns the derivative.

Result = DERIV([*X*,] *Y*)

DERIVSIG - Computes standard deviation of derivative found by DERIV.

Result = DERIVSIG([*X, Y,* Sig_x,] Sig_y)

DETERM - Computes the determinant of a square matrix.

Result = DETERM(*A* [, /CHECK] [, /DOUBLE] [, ZERO=*value*])

DEVICE - Sets to plot in device coordinates.

Note: Each keyword to DEVICE is followed by the device(s) to which it applies.

DEVICE [, /AVANTGARDE | , /BKMAN | , /COURIER | , /HELVETICA | , /ISOLATIN1 | , /PALATINO | , /SCHOOLBOOK | , /SYMBOL | , TIMES | , ZAPFCHANCERY | , ZAPFDINGBATS {PS}] [, /AVERAGE_LINES{REGIS}] [, /BINARY | , /NCAR | , /TEXT {CGM}] [, BITS_PER_PIXEL={1 | 2 | 4 | 8}{PS}] [, /BOLD{PS}] [, /BOOK{PS}] [, /BYPASS_TRANSLATION{WIN, X}] [, /CLOSE{Z}] [, /CLOSE_DOCUMENT{PRINTER}] [, /CLOSE_FILE{CGM, HP, METAFILE, PCL, PS, REGIS, TEK}] [, /CMYK{PS}] [, /COLOR{PCL, PS}] [, COLORS=*value*{CGM, TEK}] [, COPY=*[Xsource, Ysource, cols, rows, Xdest, Ydest [, Window_index]]*{WIN, X}] [, /CURSOR_CROSSHAIR{WIN, X}] [, CURSOR_IMAGE=*value*{16-element short int vector}{WIN, X}] [, CURSOR_MASK=*value*{WIN, X}] [, /CURSOR_ORIGINAL{WIN, X}] [, CURSOR_STANDARD=*value*{WIN: arrow=32512, I-beam=32513, hourglass=32514, black cross=32515, up arrow=32516, size(NT)=32640, icon(NT)=32641, size NW-SE=32642, size NE-SW=32643, size E-W=32644, size N-S=32645}{X: one of the values in file cursorfonts.h}] [, CURSOR_XY=*[x,y]*{WIN, X}] [, /DECOMPOSED{WIN, X}] [, /DIRECT_COLOR{X}] [, EJECT={0 | 1 | 2}{HP}] [, ENCAPSULATED={0 | 1}{PS}] [, ENCODING={1 (binary) | 2 (text) | 3 (NCAR binary)}{CGM}] [, FILENAME=*filename*{CGM, HP, METAFILE, PCL, PS, REGIS, TEK}] [, /FLOYD{PCL, X}] [, FONT_INDEX=*integer*{PS}] [, FONT_SIZE=*points*{PS}] [, GET_CURRENT_FONT=*variable*{METAFILE, PRINTER, WIN, X}] [, GET_DECOMPOSED=*variable*{WIN, X}] [, GET_FONTNAMES=*variable*{METAFILE, PRINTER, WIN, X}] [, GET_FONTNUM=*variable*{METAFILE, PRINTER, WIN, X}] [, GET_GRAPHICS_FUNCTION=*variable*{WIN, X, Z}] [, GET_PAGE_SIZE=*variable*{PRINTER}] [, GET_SCREEN_SIZE=*variable*{WIN, X}] [, GET_VISUAL_DEPTH=*variable*{WIN, X}] [, GET_VISUAL_NAME=*variable*{WIN, X}] [, GET_WINDOW_POSITION=*variable*{WIN, X}] [, GET_WRITE_MASK=*variable*{X, Z}] [, GIN_CHARS=*number_of_characters*{TEK}] [, GLYPH_CACHE=*number_of_glyphs*{METAFILE, PRINTER, PS, WIN, Z}] [, /INCHES{HP, METAFILE, PCL, PRINTER, PS}] [, /INDEX_COLOR{METAFILE, PRINTER}] [, /ITALIC{PS}] [, /LANDSCAPE | , /PORTRAIT{HP, PCL, PRINTER, PS}] [, LANGUAGE_LEVEL={1 | 2}{PS}] [, /DEMI | , /LIGHT | , /MEDIUM | , /NARROW | , /OBLIQUE{PS}] [, OPTIMIZE={0 | 1 | 2}{PCL}] [, /ORDERED{PCL, X}] [, OUTPUT=*scalar string*{HP, PS}] [, /PIXELS{PCL}] [, PLOT_TO=*logical unit num*{REGIS, TEK}] [, /PLOTTER_ON_OFF{HP}] [, /POLYFILL{HP}] [, PRE_DEPTH=*value*{PS}] [, PRE_XSIZE=*width*{PS}] [, PRE_YSIZE=*height*{PS}] [, /PREVIEW{PS}] [, PRINT_FILE=*filename*{WIN}] [, /PSEUDO_COLOR{X}] [, RESET_STRING=*string*{TEK}] [, RESOLUTION=*value*{PCL}] [, RETAIN={0 | 1 | 2}{WIN, X}] [, SCALE_FACTOR=*value*{PRINTER, PS}] [, SET_CHARACTER_SIZE=*[font size, line spacing]*{CGM, HP, METAFILE, PCL, PS, REGIS, TEK, WIN, X, Z}] [, SET_COLORMAP=*value*{14739-element byte vector}{PCL}] [, SET_COLORS=*value*{2 to 256}{Z}]

DEVICE - *continued*

[, SET_FONT=*scalar string*{METAFILE, PRINTER, PS, WIN, Z}] [, SET_GRAPHICS_FUNCTION=*code*{0 to 15}{WIN, X, Z}] [, SET_RESOLUTION=*[width, height]*{Z}] [, SET_STRING=*string*{TEK}] [, SET_TRANSLATION=*variable*{X}][, SET_WRITE_MASK=*value*{0 to 2^n-1 for *n*-bit system}{X, Z}] [, STATIC_COLOR=*value*{bits per pixel}{X}] [, STATIC_GRAY=*value*{bits per pixel}{X}] [, /TEK4014{TEK}] [, TEK4100{TEK}] [, THRESHOLD=*value*{PCL, X}] [, TRANSLATION=*variable*{WIN, X}] [, TRUE_COLOR=*value*{bits per pixel}{METAFILE, PRINTER, X}] [, /TT_FONT{METAFILE, PRINTER, WIN, X, Z}] [, /TTY{REGIS, TEK}] [, /VT240 | , /VT241 | , /VT340 | , /VT341 {REGIS}] [, WINDOW_STATE=*variable*{WIN, X}] [, XOFFSET=*value*{HP, PCL, PRINTER, PS}] [, XON_XOFF={0 | 1 (default)}{HP}] [, XSIZE=*width*{HP, PCL, METAFILE, PRINTER, PS}] [, YOFFSET=*value*{HP, PCL, PRINTER, PS}] [, YSIZE=*height*{HP, PCL, METAFILE, PRINTER, PS}] [, Z_BUFFERING={0 | 1 (default)}{Z}]

DFPMIN - Minimizes a function using Davidon-Fletcher-Powell method.

DFPMIN, *X*, *Gtol*, *Fmin*, *Func*, *Dfunc* [, /DOUBLE] [, EPS=*value*] [, ITER=*variable*] [, ITMAX=*value*] [, STEPMAX=*value*] [, TOLX=*value*]

DIAG_MATRIX - Constructs a diagonal matrix from an input vector, or if given a matrix, extracts a diagonal vector.

Result = DIAG_MATRIX(*A* [, Diag])

DIALOG_MESSAGE - Creates modal message dialog.

Result = DIALOG_MESSAGE(*Message_Text* [, /CANCEL] [, /CENTER] [, /DEFAULT_CANCEL | , /DEFAULT_NO] [, DIALOG_PARENT=*widget_id*] [, DISPLAY_NAME=*string*] [, /ERROR | , /INFORMATION | , /QUESTION] [, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_PICKFILE - Creates native file-selection dialog.

Result = DIALOG_PICKFILE([, DEFAULT_EXTENSION=*string*] [, /DIRECTORY] [, DIALOG_PARENT=*widget_id*] [, DISPLAY_NAME=*string*] [, FILE=*string*] [, FILTER=*string/string array*] [, /FIX_FILTER] [, GET_PATH=*variable*] [, GROUP=*widget_id*] [, /MULTIPLE_FILES] [, /MUST_EXIST] [, /OVERWRITE_PROMPT] [, PATH=*string*] [, /READ | , /WRITE] [, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_PRINTERSETUP - Opens native dialog used to set properties for a printer.

Result = DIALOG_PRINTERSETUP([*PrintDestination*] [, DIALOG_PARENT=*widget_id*] [, DISPLAY_NAME=*string*] [, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_PRINTJOB - Opens native dialog used to set parameters for a print job.

Result = DIALOG_PRINTJOB([*PrintDestination*] [, DIALOG_PARENT=*widget_id*] [, DISPLAY_NAME=*string*] [, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_READ_IMAGE - Presents GUI for reading image files.

Result = DIALOG_READ_IMAGE ([*Filename*] [, DIALOG_PARENT=*widget_id*] [, FILE=*variable*] [, FILTER_TYPE=*string*] [, /FIX_FILTER] [, GET_PATH=*variable*] [, IMAGE=*variable*] [, PATH=*string*] [, QUERY=*variable*] [, RED=*variable*] [, GREEN=*variable*] [, BLUE=*variable*] [,TITLE=*string*])

DIALOG_WRITE_IMAGE - Presents GUI for writing image files.

Result = DIALOG_WRITE_IMAGE (*Image* [, *R*, *G*, *B*] [, DIALOG_PARENT=*widget_id*] [, FILE=*string*] [, /FIX_TYPE] [, /NOWRITE] [, OPTIONS=*variable*] [, PATH=*string*] [,TITLE=*string*] [, TYPE=*variable*] [, /WARN_EXIST])

DICOMEX_GETCONFIGFILEPATH - See Appendix B, "IDL DICOM Quick Reference" in the *Medical Imaging in IDL* manual.

DICOMEX_GETSTORSCPDIR - See Appendix B, "IDL DICOM Quick Reference" in the *Medical Imaging in IDL* manual.

DICOMEX_NET - See Appendix B, "IDL DICOM Quick Reference" in the *Medical Imaging in IDL* manual.

DIGITAL_FILTER - Calculates coefficients of a non-recursive, digital filter.

Result = DIGITAL_FILTER(*Flow*, *Fhigh*, *A*, *Nterms* [, /DOUBLE])

DILATE - Implements morphologic dilation operator on binary and grayscale images.

Result = DILATE(*Image*, *Structure* [, X_0 [, Y_0 [, Z_0]]] [, /CONSTRAINED [, BACKGROUND=*value*]] [, /GRAY [, /PRESERVE_TYPE | , /UINT | , /ULONG]] [, VALUES=*array*])

DINDGEN - Returns a double-precision array with each element set to its subscript.

Result = DINDGEN(D_1 *[, ...,*D_8*]* [, *Thread pool keywords*])

DISSOLVE - Provides a digital "dissolve" effect for images.

DISSOLVE, *Image* [, DELAY=*seconds*] [, /ORDER] [, SIZ=*pixels*] [, X0=*pixels*, Y0=*pixels*]

DIST - Creates array with each element proportional to its frequency.

Result = DIST(*N* [, *M*])

DISTANCE_MEASURE - Computes the pairwise distance between a set of items or observations.
Result = DISTANCE_MEASURE(*Array* [, /DOUBLE] [, /MATRIX] [, MEASURE=*value*] [, POWER_MEASURE=*value*])

DLM_LOAD - Explicitly causes a DLM to be loaded.
DLM_LOAD, $DLMNameStr_1$ [, $DLMNameStr_2$,..., $DLMNameStr_n$]

DOC_LIBRARY - Extracts documentation headers from IDL programs.
DOC_LIBRARY [, *Name*] [, /PRINT]
UNIX keywords: [, DIRECTORY=*string*] [, /MULTI]

DOUBLE - Converts argument to double-precision type.
Result = DOUBLE(*Expression [, Offset [, D_1 [, ...,D_8]]]* [, *Thread pool keywords*])

DRAW_ROI - Draws region or group of regions to current Direct Graphics device.
DRAW_ROI, *oROI* [, /LINE_FILL] [, SPACING=*value*]
Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP] [, ORIENTATION=*ccw_degrees_from_horiz*] [, PSYM=*integer*{0 to 10}] [, SYMSIZE=*value*] [, /T3D] [, THICK=*value*]

E

EFONT - Interactive vector font editor and display tool.
EFONT, *Init_Font* [, /BLOCK] [, GROUP=*widget_id*]

EIGENQL - Computes eigenvalues and eigenvectors of a real, symmetric array.
Result = EIGENQL(*A* [, /ABSOLUTE] [, /ASCENDING] [, /DOUBLE] [, EIGENVECTORS=*variable*] [, /OVERWRITE | , RESIDUAL=*variable*])

EIGENVEC - Computes eigenvectors of a real, non-symmetric array.
Result = EIGENVEC(*A, Eval* [, /DOUBLE] [, ITMAX=*value*] [, RESIDUAL=*variable*])

ELMHES - Reduces nonsymmetric array to upper Hessenberg form.
Result = ELMHES(*A* [, /COLUMN] [, /DOUBLE] [, /NO_BALANCE])

EMPTY - Empties the graphics output buffer.
EMPTY

ENABLE_SYSRTN - Enables/disables IDL system routines.
ENABLE_SYSRTN [, *Routines*] [, /DISABLE] [, /EXCLUSIVE] [, /FUNCTIONS]

EOF - Tests the specified file for the end-of-file condition.
Result = EOF(*Unit*)

EOS_* Routines - See "EOS Routines" on page 127.

.ERASE - Erases the screen of the current graphics device, or starts a new page if the device is a printer.
ERASE [, *Background_Color*] [, CHANNEL=*value*] [, COLOR=*value*]

ERODE - Implements the erosion operator on binary and grayscale images and vectors.
Result = ERODE(*Image, Structure* [, X_0 [, Y_0 [, Z_0]]] [, /GRAY [, /PRESERVE_TYPE | , /UINT | , /ULONG]] [, VALUES=*array*])

ERF - Returns the value of an error function.
Result = ERF(*Z* [, *Thread pool keywords*])

ERFC - Returns the value of a complementary error function.
Result = ERFC(*Z* [, *Thread pool keywords*])

ERFCX - Returns the value of a scaled complementary error function.
Result = ERFCX(*Z* [, *Thread pool keywords*])

ERRPLOT - Plots error bars over a previously drawn plot.
ERRPLOT, [*X,*] *Low, High* [, WIDTH=*value*]

EXECUTE - Compiles and executes IDL statements contained in a string.
Result = EXECdUTE(*String* [, *QuietCompile*] [, *QuietExecution*])

EXIT - Quits IDL and exits back to the operating system.
EXIT [, /NO_CONFIRM] [, STATUS=*code*]

EXP - Returns the natural exponential function of *Expression.*
Result = EXP(*Expression* [, *Thread pool keywords*])

EXPAND - Shrinks/expands image using bilinear interpolation.
EXPAND, *A, Nx, Ny, Result* [, FILLVAL=*value*] [, MAXVAL=*value*]

EXPAND_PATH - Expands path-definition string into full path name for use with the !PATH system variable.
Result = EXPAND_PATH(*String* [, /ALL_DIRS] [, /ARRAY] [, COUNT=*variable*] [, /DLM] [, /HELP])

EXPINT - Returns the value of the exponential integral.
Result = EXPINT(*N, X* [, /DOUBLE] [, EPS=*value*] [,ITER=*variable*] [, ITMAX=*value*] [, *Thread pool keywords*])

EXTRAC - Returns sub-matrix of input array. Array operators (e.g., * and :) should usually be used instead.
Result = EXTRAC(*Array,* C_1, C_2, ..., C_n, S_1, S_2, ..., S_n)

EXTRACT_SLICE - Returns 2D planar slice extracted from volume.
Result = EXTRACT_SLICE(*Vol*, *Xsize*, *Ysize*, *Xcenter*, *Ycenter*, *Zcenter*, *Xrot*, *Yrot*, *Zrot*
[, ANISOTROPY=*[xspacing, yspacing, zspacing]*]
[, /CUBIC] [, OUT_VAL=*value*] [, /RADIANS]
[, /SAMPLE] [, VERTICES=*variable*])
or
Result = EXTRACT_SLICE(*Vol*, *Xsize*, *Ysize*, *Xcenter*, *Ycenter*, *Zcenter*, *PlaneNormal*, *Xvec*
[, ANISOTROPY=*[xspacing, yspacing, zspacing]*]
[, /CUBIC] [, OUT_VAL=*value*] [, /RADIANS]
[, /SAMPLE] [, VERTICES=*variable*])

F

F_CVF - Computes the cutoff value in an F distribution.
Result = F_CVF(*P, Dfn, Dfd*)

F_PDF - Computes the F distribution function.
Result = F_PDF(*V, Dfn, Dfd*)

FACTORIAL - Computes the factorial *N*!.
Result = FACTORIAL(*N* [, /STIRLING] [, /UL64])

FFT - Returns the Fast Fourier Transform of *Array*.
Result = FFT(*Array* [, *Direction*] [, DIMENSION=*vector*]
[, /DOUBLE] [, /INVERSE] [, /OVERWRITE]
[, *Thread pool keywords*])

FILE_BASENAME - Returns the *basename* of a *file path*.
Result = FILE_BASENAME(*Path* [, *RemoveSuffix*]
[, /FOLD_CASE])

FILE_CHMOD - Changes the current access permissions (or modes) associated with a file or directory.
FILE_CHMOD, *File* [, *Mode*]
[, /A_EXECUTE | /A_READ | , /A_WRITE]
[, /G_EXECUTE | /G_READ | , /G_WRITE]
[, /O_EXECUTE | /O_READ | , /O_WRITE]
[, /NOEXPAND_PATH]
[, /U_EXECUTE | /U_READ | , /U_WRITE]
UNIX-Only Keywords: [, /SETGID] [, /SETUID]
[, /STICKY_BIT]

FILE_COPY - Copies files or directories to a new location.
FILE_COPY, *SourcePath*, *DestPath* [, /ALLOW_SAME]
[, /NOEXPAND_PATH] [, /OVERWRITE]
[, /RECURSIVE] [, /REQUIRE_DIRECTORY]
[, /VERBOSE]
Unix-Only Keywords: [, /COPY_NAMED_PIPE]
[, /COPY_SYMLINK] [, /FORCE]

FILE_DELETE - Deletes a file or empty directory, if the process has the necessary permissions to remove the file as defined by the current operating system.
FILE_DELETE, *File1* [,... *FileN*]
[, /ALLOW_NONEXISTENT] [, /NOEXPAND_PATH]
[, /QUIET] [, /RECURSIVE] [, VERBOSE]

FILE_DIRNAME - Returns the *dirname* of a *file path*.
Result = FILE_DIRNAME(*Path*
[, /MARK_DIRECTORY])

FILE_EXPAND_PATH - Expands a given file or partial directory name to its fully qualified name regardless of the current working directory.
Return = FILE_EXPAND_PATH (*Path*)

FILE_INFO - Returns status information about a file.
Result = FILE_INFO(*Path* [, /NOEXPAND_PATH])

FILE_LINES - Returns the number of lines of text in a file.
FILE_LINES(*Path* [,/COMPRESS]
[, /NOEXPAND_PATH])

FILE_LINK - Creates UNIX file links.
FILE_LINK, *SourcePath*, *DestPath* [, /ALLOW_SAME]
[, /HARDLINK] [, /NOEXPAND_PATH] [, /VERBOSE]

FILE_MKDIR - Creates a new directory, or directories, with default access permissions for the current process.
FILE_MKDIR, *File1* [,... *FileN*] [, /NOEXPAND_PATH]

FILE_MOVE - Renames files and directories.
FILE_MOVE, *SourcePath*, *DestPath* [, /ALLOW_SAME]
[, /NOEXPAND_PATH] [, /OVERWRITE]
[, /REQUIRE_DIRECTORY] [, /VERBOSE]

FILE_POLL_INPUT - Blocks processing until it detects that a read operation on a specified file will succeed.
Result = FILE_POLL_INPUT(*Units* [, COUNT=*variable*]
[, TIMEOUT=*value*])

FILE_READLINK - Returns the path pointed to by a UNIX symbolic link.
FILE_READLINK(*Path* [, /ALLOW_NONEXISTENT]
[, /ALLOW_NONSYMLINK] [, /NOEXPAND_PATH])

FILE_SAME - Determines whether two different file names refer to the same underlying file.
Result = FILE_SAME(*Path1*, *Path2*
[, /NOEXPAND_PATH])

FILE_SEARCH - Returns a string array containing the names of all files matching the input path specification.

Result = FILE_SEARCH(*Path_Specification*)

or

Result = FILE_SEARCH(*Dir_Specification*, *Recur_Pattern*)

Keywords: [, COUNT=*variable*] [, /EXPAND_ENVIRONMENT] [, /EXPAND_TILDE] [, /FOLD_CASE] [, /FULLY_QUALIFY_PATH] [, /ISSUE_ACCESS_ERROR] [, /MARK_DIRECTORY] [, /MATCH_ALL_INITIAL_DOT | /MATCH_INITIAL_DOT] [, /NOSORT] [, /QUOTE] [, /TEST_DIRECTORY] [, /TEST_EXECUTABLE] [, /TEST_READ] [, /TEST_REGULAR] [, /TEST_WRITE] [, /TEST_ZERO_LENGTH] [, /WINDOWS_SHORT_NAMES]

UNIX-Only Keywords: [, /TEST_BLOCK_SPECIAL] [, /TEST_CHARACTER_SPECIAL] [, /TEST_DANGLING_SYMLINK] [, /TEST_GROUP] [, /TEST_NAMED_PIPE] [, /TEST_SETGID] [, /TEST_SETUID] [, /TEST_SOCKET] [, /TEST_STICKY_BIT] [, /TEST_SYMLINK] [, /TEST_USER]

FILE_TEST - Checks files for existence and other file attributes without first having to open the file.

Result = FILE_TEST(*File* [, /DIRECTORY | , /EXECUTABLE | , /READ | , /REGULAR | , /WRITE | , /ZERO_LENGTH] [, GET_MODE=*variable*] [, /NOEXPAND_PATH])

UNIX-Only Keywords: [, /BLOCK_SPECIAL | , /CHARACTER_SPECIAL | , /DANGLING_SYMLINK | , /GROUP | , /NAMED_PIPE | , /SETGID | , /SETUID | , /SOCKET | , /STICKY_BIT | , /SYMLINK | , /USER]

FILE_WHICH - Separates a specified file path into its component directories, and searches each directory in turn for a specific file.

Result = FILE_WHICH([*Path*,] *File* [, /INCLUDE_CURRENT_DIR])

FILEPATH - Returns full path to a file in the IDL distribution.

Result = FILEPATH(*Filename* [, ROOT_DIR=*string*] [, SUBDIRECTORY=*string/string_array*] [, /TERMINAL] [, /TMP])

FINDGEN - Returns a floating-point array with each element set to its subscript.

Result = FINDGEN(D_1 *[, ...,*D_8*]* [, *Thread pool keywords*])

FINITE - Returns True if its argument is finite.

Result = FINITE(*X* [, /INFINITY] [, /NAN] [, SIGN=*value*] [, *Thread pool keywords*])

FIX - Converts argument to integer type, or type specified by TYPE keyword.

Result = FIX(*Expression [, Offset [,* D_1 *[, ...,*D_8*]]]* [, /PRINT] [, TYPE=*type code*{0 to 15}] [, *Thread pool keywords*])

FLICK - Causes the display to flicker between two images.

FLICK, *A, B* [, *Rate*]

FLOAT - Converts argument to single-precision floating-point.

Result = FLOAT(*Expression [, Offset [,* D_1 *[, ...,*D_8*]]]* [, *Thread pool keywords*])

FLOOR - Returns closest integer less than or equal to argument.

Result = FLOOR(*X* [, /L64] [, *Thread pool keywords*])

FLOW3 - Draws lines representing a 3D flow/velocity field.

FLOW3, *Vx, Vy, Vz* [, ARROWSIZE=*value*] [, /BLOB] [, LEN=*value*] [, NSTEPS=*value*] [, NVECS=*value*] [, SX=*vector*, SY=*vector*, SZ=*vector*]

FLTARR - Returns a single-precision, floating-point vector or array.

Result = FLTARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

FLUSH - Flushes file unit buffers.

FLUSH, $Unit_1$, ..., $Unit_n$

FOR - Executes statements repeatedly, incrementing or decrementing a variable with each repetition, until a condition is met.

FOR *variable = init, limit* [, *Increment*] DO *statement*

or

FOR *variable = init, limit* [, *Increment*] DO BEGIN
statements
ENDFOR

FORMAT_AXIS_VALUES - Formats numbers as strings for use as axis values.

Result = FORMAT_AXIS_VALUES(*Values*)

FORWARD_FUNCTION - Causes argument(s) to be interpreted as functions rather than variables (versions of IDL prior to 5.0 used parentheses to declare arrays).

FORWARD_FUNCTION $Name_1$, $Name_2$, ..., $Name_n$

FREE_LUN - Frees previously-reserved file units.

FREE_LUN [, $Unit_1$, ..., $Unit_n$] [, EXIT_STATUS=*variable*] [, /FORCE]

FSTAT - Returns information about a specified file unit.

Result = FSTAT(*Unit*)

FULSTR - Restores a sparse matrix to full storage mode.

Result = FULSTR(*A*)

FUNCT - Evaluates sum of a Gaussian and a 2nd-order polynomial and returns value of its partial derivatives.

FUNCT, *X, A, F* [, *Pder*]

FUNCTION - Defines a function.

FUNCTION *Function_Name*, $parameter_1$, ..., $parameter_n$

FV_TEST - Performs the F-variance test.

Result = FV_TEST(*X, Y*)

FX_ROOT - Computes real and complex roots of a univariate nonlinear function using an optimal Müller's method.
Result = FX_ROOT(*X*, *Func* [, /DOUBLE] [, ITMAX=*value*] [, /STOP] [, TOL=*value*])

FZ_ROOTS - Finds the roots of a complex polynomial using Laguerre's method.
Result = FZ_ROOTS(*C* [, /DOUBLE] [, EPS=*value*] [, /NO_POLISH])

G

GAMMA - Returns the gamma function of *Z*.
Result = GAMMA(*Z* [, *Thread pool keywords*])

GAMMA_CT - Applies gamma correction to a color table.
GAMMA_CT, *Gamma* [, /CURRENT] [, /INTENSITY]

GAUSS_CVF - Computes cutoff value in Gaussian distribution.
Result = GAUSS_CVF(*P*)

GAUSS_PDF - Computes Gaussian distribution function.
Result = GAUSS_PDF(*V*)

GAUSS2DFIT - Fits a 2D elliptical Gaussian equation to rectilinearly gridded data.
Result = GAUSS2DFIT(*Z*, *A* [, *X*, *Y*] [, /NEGATIVE] [, /TILT])

GAUSSFIT - Fits the sum of a Gaussian and a quadratic.
Result = GAUSSFIT(*X*, *Y* [, *A*] [, CHISQ=*variable*] [, ESTIMATES=*array*] [, MEASURE_ERRORS=*vector*] [, NTERMS=*integer*{3 to 6}] [, SIGMA=*variable*] [, YERROR=*variable*])

GAUSSINT - Returns integral of Gaussian probability function.
Result = GAUSSINT(*X* [, *Thread pool keywords*])

GET_DRIVE_LIST - Returns string array of the names of valid drives/volumes for the file system.
Result = GET_DRIVE_LIST([, COUNT=*variable*])
Windows keywords: [, /CDROM] [, /FIXED] [, /REMOTE] [, /REMOVABLE]

GET_KBRD - Gets one input character.
Result = GET_KBRD([*Wait*] [,/ESCAPE | ,/KEY_NAME])

GET_LUN - Reserves a logical unit number (file unit).
GET_LUN, *Unit*

GET_SCREEN_SIZE - Returns dimensions of the screen.
Result = GET_SCREEN_SIZE([*Display_name*] [, RESOLUTION=*variable*])
X Windows Keywords: [, DISPLAY_NAME=*string*]

GETENV - Returns the value of an environment variable.
Result = GETENV(*Name* [, /ENVIRONMENT])

GOTO - Transfers program control to point specified by *label*.
GOTO, *label*

GRID_INPUT - Preprocesses and sorts two-dimensional scattered data points, and removes duplicate values.
GRID_INPUT, *X*, *Y*, *F*, *X1*, *Y1*, *F1* [, DUPLICATES=*string*] [, EPSILON=*value*] [, EXCLUDE=*vector*]
or
GRID_INPUT, *Lon*, *Lat*, *F*, *Xyz*, *F1*, /SPHERE [, /DEGREES] [, DUPLICATES=*string*] [, EPSILON=*value*] [, EXCLUDE=*vector*]
or
GRID_INPUT, *R*, *Theta*, *F*, *X1*, *Y1*, *F1*, /POLAR [, /DEGREES] [, DUPLICATES=*string*] [, EPSILON=*value*] [, EXCLUDE=*vector*]

GRID_TPS - Uses thin plate splines to interpolate a set of values over a regular 2D grid, from irregularly sampled data values.
Interp = GRID_TPS (*Xp*, *Yp*, *Values* [, COEFFICIENTS=*variable*] [, NGRID=*[nx, ny]*] [, START=*[x0, y0]*] [, DELTA=*[dx, dy]*])

GRID3 - Creates a regularly-gridded 3D dataset from a set of scattered 3D nodes.
Result = GRID3(*X*, *Y*, *Z*, *F*, *Gx*, *Gy*, *Gz* [, DELTA=*scalar/vector*] [, DTOL=*value*] [, GRID=*value*] [, NGRID=*value*] [, START=*[x, y, z]*])

GRIDDATA - Interpolates scattered data values and locations sampled on a plane or a sphere to a regular grid.
Result = GRIDDATA(*X*, *F*)
or
Result = GRIDDATA(*X*, *Y*, *F*)
or
Result = GRIDDATA(*X*, *Y*, *Z*, *F*, /SPHERE)
or
Result = GRIDDATA(*Lon*, *Lat*, *F*, /SPHERE)

GS_ITER - Solves linear system using Gauss-Seidel iteration.
Result = GS_ITER(*A*, *B* [, /CHECK] [, /DOUBLE] [, LAMBDA=*value*{0.0 to 2.0}] [, MAX_ITER=*value*] [, TOL=*value*] [, X_0=*vector*])

H

H_EQ_CT - Histogram-equalizes the color tables for an image or a region of the display.
H_EQ_CT [, *Image*]

H_EQ_INT - Interactively histogram-equalizes the color tables of an image or a region of the display.
H_EQ_INT [, *Image*]

HANNING - Creates Hanning and Hamming windows.
Result = HANNING(N_1 [, N_2] [, ALPHA=*value*{0.5 to 1.0}] [, /DOUBLE])

HDF_* Routines - See "HDF Routines" on page 132t.

HDF5_* Routines - See "HDF5 Routines" on page 137.

HDF_BROWSER - Opens GUI to view contents of HDF, HDF-EOS, or NetCDF file.

Template = HDF_BROWSER([*Filename*] [, CANCEL=*variable*] [, GROUP=*widget_id*] [, PREFIX=*string*])

HDF_READ - Extracts HDF, HDF-EOS, and NetCDF data and metadata into an output structure.

Result = HDF_READ([*Filename*] [, DFR8=*variable*] [, DF24=*variable*] [, PREFIX=*string*] [, TEMPLATE=*value*])

HEAP_FREE - Recursively frees all heap variables referenced by its input argument.

HEAP_FREE, *Var* [, /OBJ] [, /PTR] [, /VERBOSE]

HEAP_GC - Performs garbage collection on heap variables.

HEAP_GC [, /OBJ | , /PTR] [, /VERBOSE]

HEAP_NOSAVE - Used to clear the *save attribute* of pointer or object heap variables.

HEAP_NOSAVE, *HeapVar*

HEAP_SAVE - Used to query whether a pointer or object heap variable is savable. It can also be used to change the heap variable save attribute.

Result = HEAP_SAVE(*HeapVar* [, *Set*])

HELP - Provides information about the current IDL session.

HELP, $Expression_1$, ..., $Expression_n$ [, /BREAKPOINTS] [, /BRIEF] [, /DEVICE] [, /DLM] [, /FILES] [, /FULL] [, /FUNCTIONS] [, /HEAP_VARIABLES] [, /KEYS] [, /LAST_MESSAGE] [, LEVEL=*value*] [, /MEMORY] [, /MESSAGES] [, NAMES=*string_of_variable_names*] [, /OBJECTS] [, OUTPUT=*variable*] [, /PATH_CACHE] [, /PREFERENCES] [, /PROCEDURES] [, /RECALL_COMMANDS] [, /ROUTINES] [, /SHARED_MEMORY] [, /SOURCE_FILES] [, /STRUCTURES] [, /SYSTEM_VARIABLES] [, /TRACEBACK]

HILBERT - Constructs a Hilbert transform.

Result = HILBERT(*X* [, *D*])

HIST_2D - Returns histogram of two variables.

Result = HIST_2D(V_1, V_2 [, BIN1=*width*] [, BIN2=*height*] [, MAX1=*value*] [, MAX2=*value*] [, MIN1=*value*] [, MIN2=*value*])

HIST_EQUAL - Histogram-equalizes an image.

Result = HIST_EQUAL(*A* [, BINSIZE=*value*] [, FCN=*vector*] [, /HISTOGRAM_ONLY] [, MAXV=*value*] [, MINV=*value*] [, OMAX=*variable*] [, OMIN=*variable*] [, PERCENT=*value*] [, TOP=*value*])

HISTOGRAM - Computes the density function of an array.

Result = HISTOGRAM(*Array* [, BINSIZE=*value*] [, INPUT=*variable*] [, LOCATIONS=*variable*] [, MAX=*value*] [, MIN=*value*] [, /NAN] [, NBINS=*value*] [, OMAX=*variable*] [, OMIN=*variable*] [, /L64 | REVERSE_INDICES=*variable*])

HLS - Creates color table in Hue, Lightness, Saturation color system.

HLS, *Litlo*, *Lithi*, *Satlo*, *Sathi*, *Hue*, *Loops* [, *Colr*]

HOUGH - Returns the Hough transform of a two-dimensional image.

Hough Transform: *Result* = HOUGH(*Array* [, /DOUBLE] [, DRHO=*scalar*] [, DX=*scalar*] [, DY=*scalar*] [, /GRAY] [, NRHO=*scalar*] [, NTHETA=*scalar*] [, RHO=*variable*] [, RMIN=*scalar*] [, THETA=*variable*] [, XMIN=*scalar*] [, YMIN=*scalar*])

Hough Backprojection: *Result* = HOUGH(*Array*, /BACKPROJECT, RHO=*variable*, THETA=*variable* [, /DOUBLE] [, DX=*scalar*] [, DY=*scalar*] [, NX=*scalar*] [, NY=*scalar*] [, XMIN=*scalar*] [, YMIN=*scalar*])

HQR - Returns all eigenvalues of an upper Hessenberg array.

Result = HQR(*A* [, /COLUMN] [, /DOUBLE])

HSV - Creates color table based on Hue/Saturation Value color system.

HSV, *Vlo*, *Vhi*, *Satlo*, *Sathi*, *Hue*, *Loops* [, *Colr*]

I

IBETA - Computes the incomplete beta function.

Result = IBETA(*A*, *B*, *Z*[, /DOUBLE] [, EPS=*value*] [, ITER=*variable*] [, ITMAX=*value*])

ICONTOUR - Creates an iTool and associated user interface (UI) configured to display and manipulate contour data.

ICONTOUR[, *Z*[, *X*, *Y*]]

iTool Common Keywords:

[, BACKGROUND_COLOR=*value*]
[, DIMENSIONS=[*x*, *y*]]
[, /DISABLE_SPLASH_SCREEN]
[, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]]
[, MACRO_NAMES=*string or string array*]
[, NAME=*string*] [. /NO_SAVEPROMPT]
[, OVERPLOT=*iToolID*] {, STYLE_NAME=*string*]
[, TITLE=*string*] [, VIEW_GRID=[*columns*, *rows*]]
[, /VIEW_NEXT] [, VIEW_NUMBER=*integer*]
[, VIEW_TITLE=*string*]

iTool Contour Keywords: [, RGB_INDICES=*vector of indices*] [, RGB_TABLE=*byte array of 256 by 3 or 3 by 256 elements*] [, ZVALUE=*value*]

Contour Object Keywords: [, AM_PM=*vector of two strings*] [, ANISOTROPY=[*x, y, z*]] [, C_COLOR=*color array*] [, C_FILL_PATTERN=*array of IDLgrPattern objects*] [, C_LABEL_INTERVAL=*vector*]
[, C_LABEL_NOGAPS=*vector*]
[, C_LABEL_OBJECTS=*array of object references*]
[, C_LABEL_SHOW=*vector of integers*]
[, C_LINESTYLE=*array of linestyles*]
[, C_THICK=*float array*{each element 1.0 to 10.0}]
[, C_USE_LABEL_COLOR=*vector of values*]
[, C_USE_LABEL_ORIENTATION=vector of values]
[, C_VALUE=*scalar or vector*] [, CLIP_PLANES=*array*]
[, COLOR=*RGB vector*] [, DAYS_OF_WEEK=*vector of seven strings*] [, DEPTH_OFFSET=*value*]
[, /DOWNHILL] [, /FILL] [, GRID_UNITS=*value*]
[, /HIDE] [, LABEL_FONT=*objref*]
[, LABEL_FORMAT=*string*]
[, LABEL_FRMTDATA=*value*]
[, LABEL_UNITS=*string*] [, MAX_VALUE=*value*]
[, MIN_VALUE=*value*] [, MONTHS=*vector of 12 values*]
[, N_LEVELS=*value*] [, /PLANAR]
[, SHADE_RANGE=[*min, max*]] [, SHADING={0 |1}]
[, TICKINTERVAL=*value*] [, TICKLEN=*value*]
[, USE_TEXT_ALIGNMENTS=*value*]

Axis Object Keywords: [, {X | Y | Z}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]
[, {X | Y | Z}MAJOR=*integer*]
[, {X | Y | Z}MINOR=*integer*]
[, {X | Y | Z}RANGE=[*min, max*]] [, {X | Y | Z} SUBTICKLEN=*ratio*]
[, {X | Y | Z}TEXT_COLOR=*RGB vector*]
[, {X | Y | Z}TICKFONT_INDEX= {0 | 1 | 2 | 3 | 4}]
[, {X | Y | Z}TICKFONT_SIZE=*integer*] [, {X | Y | Z}
[, TICKFONT_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z} TICKFORMAT=*string or string array*] [, {X | Y | Z} TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT={0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z} TICKNAME=*string array*] [, {X | Y | Z}TICKUNITS=*string*] [, {X | Y | Z}TICKVALUES=*vector*]
[, {X | Y | Z} TITLE=*string*]

IDENTITY - Returns an identity array.

Result = IDENTITY(*N* [, /DOUBLE])

IDL_Container Object - See "IDL_Container" on page 75.

IDL_VALIDNAME - Determines whether a string may be used as a valid IDL variable name or structure tag name.

Result = IDL_VALIDNAME(*String*[, /CONVERT_ALL] [, CONVERT_SPACES])

IDLanROI Object - See "IDLanROI" on page 76.

IDLanROIGroup Object - See "IDLanROIGroup" on page 76.

IDLffDICOM Object - See "IDLffDICOM" on page 77.

IDLffDicomEx Object - See Appendix B, "IDL DICOM Quick Reference" in the *Medical Imaging in IDL* manual.

IDLffDXF Object - See "IDLffDXF" on page 78.

IDLffJPEG2000 Object - See "IDLffJPEG2000" on page 78.

IDLffLangCat Object - See "IDLffLangCat" on page 79.

IDLffMrSID Object - See "IDLffMrSID" on page 79.

IDLffShape Object - See "IDLffShape" on page 79.

IDLffXMLDOM* - See "IDLffXMLDOM Classes" on page 80.

IDLffXMLSAX Object - See "IDLffXMLSAX" on page 85.

IDLgr* Objects - IDLgr* objects and their methods are described starting with "IDLgrAxis" on page 86.

IDLit* Objects - IDLit* objects and their methods are described starting with "IDLitCommand" on page 100.

IDLITSYS_CREATETOOL - Creates an instance of the specified tool registered within the iTools system.

Result = IDLITSYS_CREATETOOL(*StrTool*
[, DIMENSIONS=[*width,height*]]
[, /DISABLE_SPLASH_SCREEN]
[, IDENTIFIER=*variable*] [, INITIAL_DATA=*data*]
[, LOCATION=[*x, y*]]
[, MACRO_NAMES=*string or string array*]
[, /NO_SAVEPROMPT] [OVERPLOT=*iToolID*]
[, STYLE_NAME=*string*]
[, USER_INTERFACE=*string*] [, VIEW_GRID=*vector*]
[, /VIEW_NEXT] [, VIEW_NUMBER=*number*]
[, VISUALIZATION_TYPE=*vistype*])

IF...THEN...ELSE - Conditionally executes a statement or block of statements.

IF *expression* THEN *statement* [ELSE *statement*]
or
IF *expression* THEN BEGIN
statements
ENDIF [ELSE BEGIN
statements
ENDELSE]

IGAMMA - Computes the incomplete gamma function.

Result = IGAMMA(*A*, *Z* [, /DOUBLE] [, EPS=*value*] [, ITER=*variable*] [, ITMAX=*value*] [, METHOD=*variable*])

IIMAGE - Creates an iTool and associated user interface (UI) configured to display and manipulate image data.

IIMAGE[, *Image*[, *X*, *Y*]]

iTool Common Keywords:

[, BACKGROUND_COLOR=*value*] [, DIMENSIONS=[*x*, *y*]] [, /DISABLE_SPLASH_SCREEN] [, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]] [, MACRO_NAMES=*string or string array*] [, NAME=*string*] [. /NO_SAVEPROMPT] [, OVERPLOT=*iToolID*] {, STYLE_NAME=*string*] [, TITLE=*string*] [, VIEW_GRID=[*columns*, *rows*]] [, /VIEW_NEXT] [, VIEW_NUMBER=*integer*] [, VIEW_TITLE=*string*]

iTool Image Keywords: [, ALPHA_CHANNEL=*2-D array*] [, BLUE_CHANNEL=*2-D array*] [, GREEN_CHANNEL=*2-D array*] [, IMAGE_DIMENSIONS=[*width*, *height*]] [, IMAGE_LOCATION=[*x*, *y*]] [, RED_CHANNEL=*2-D array*] [, RGB_TABLE=*byte array of 256 by 3 or 3 by 256 elements*] [, ZVALUE=*value*]

Image Object Keywords:

[, BLEND_FUNCTION=*vector*] [, CHANNEL=*hexadecimal bitmask*] [, CLIP_PLANES=*array*] [, /HIDE] [, /INTERPOLATE] [, /ORDER] [, SUB_RECT=[*x*, *y*, *xdim*, *ydim*]]

IMAGE_CONT - Overlays an image with a contour plot.

IMAGE_CONT, *A* [, /ASPECT] [, /INTERP] [, /WINDOW_SCALE]

IMAGE_STATISTICS - Computes sample statistics for a given array of values.

IMAGE_STATISTICS, *Data* [, /LABELED | [, /WEIGHTED] [, WEIGHT_SUM=*variable*]] [, /VECTOR] [, LUT=*array*] [, MASK=*array*] [, COUNT=*variable*] [, MEAN=*variable*] [, STDDEV=*variable*] [, DATA_SUM=*variable*] [, SUM_OF_SQUARES=*variable*] [, MINIMUM=*variable*] [, MAXIMUM=*variable*] [, VARIANCE=*variable*]

IMAGINARY - Returns the imaginary part of a complex value.

Result = IMAGINARY(*Complex_Expression* [, *Thread pool keywords*])

IMAP - Creates an iTool and associated user interface (UI) configured to display and manipulate map data.

IMAP[, MAP_PROJECTION=*string*]

or

IMAP[, *Image*[, *X*, *Y*]] [, GRID_UNITS=*value*] [, MAP_PROJECTION=*string*]

or

IMAP[, *Z*[, *X*, *Y*]] , /CONTOUR [, GRID_UNITS=*value*] [, MAP_PROJECTION=*string*]

iTool Common Keywords:

[, BACKGROUND_COLOR=*value*] [, DIMENSIONS=[*x*, *y*]] [, /DISABLE_SPLASH_SCREEN] [, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]] [, MACRO_NAMES=*string or string array*] [, NAME=*string*] [, /NO_SAVEPROMPT] [, OVERPLOT=*iToolID*] [, STYLE_NAME=*string*] [, TITLE=*string*] [, VIEW_GRID=[*columns*, *rows*]] [, /VIEW_NEXT] [, VIEW_NUMBER=*integer*] [, VIEW_TITLE=*string*]

iTool Image Keywords: This procedure accepts all IIMAGE keywords.

iTool Contour Keywords: If the CONTOUR keyword is set, this procedure accepts all ICONTOUR keywords.

Map Projection Keywords:

[, CENTER_LATITUDE=*value*] [, CENTER_LONGITUDE=*value*] [, DATUM=*string*] [, FALSE_EASTING=*value*] [, FALSE_NORTHING=*value*] [, HEIGHT=*value*] [, HOM_AZIM_LONGITUDE=*value*] [, HOM_AZIM_ANGLE=*value*] [, HOM_LATITUDE1=*value*] [, HOM_LATITUDE2=*value*] [, HOM_LONGITUDE1=*value*] [, HOM_LONGITUDE2=*value*] [, IS_ZONES=*value*] [, IS_JUSTIFY=*value*] [, LIMIT=[*latmin*, *lonmin*, *latmax*, *lonmax*]] [, MERCATOR_SCALE=*value*] [, OEA_ANGLE=*value*] [, OEA_SHAPEM=*value*] [, OEA_SHAPEN=*value*] [, SEMIMAJOR_AXIS=*value*] [, SEMIMINOR_AXIS=*value*] [, SOM_INCLINATION=*value*] [, SOM_LONGITUDE=*value*] [, SOM_PERIOD=*value*] [, SOM_RATIO=*value*] [, SOM_FLAG=*value*] [, SOM_LANDSAT_NUMBER=*value*] [, SOM_LANDSAT_PATH=*value*] [, SPHERE_RADIUS=*value*] [, STANDARD_PARALLEL=*value*] [, STANDARD_PAR1=*value*] [, STANDARD_PAR2=*value*] [, TRUE_SCALE_LATITUDE=*value*] [, ZONE=*value*]

IMAP - *continued*

Axis Keywords:
[, [XY]GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]
[, [XY]MAJOR=*integer*] [, [XY]MINOR=*integer*]
[, [XY]RANGE=[*min, max*]]
[, [XY]SUBTICKLEN=*ratio*]
[, [XY]TEXT_COLOR=*RGB vector*]
[, [XY]TICKFONT_INDEX={0 | 1 | 2 | 3 | 4}]
[, [XY]TICKFONT_SIZE=*integer*]
[, [XY]TICKFONT_STYLE={0 | 1 | 2 | 3}]
[, [XY]TICKFORMAT=*string or string array*]
[, [XY]TICKINTERVAL=*value*]
[, [XY]TICKLAYOUT={0 | 1 | 2}]
[, [XY]TICKLEN=*value*]
[, [XY]TICKNAME=*string array*]
[, [XY]TICKUNITS=*string*]
[, [XY]TICKVALUES=*vector*] [, [XY]TITLE=*string*]

INDGEN - Return an integer array with each element set to its subscript.
Result = INDGEN(*D_1 [, ...,D_8]* [, /BYTE | , /COMPLEX | , /DCOMPLEX | , /DOUBLE | , /FLOAT | , /L64 | , /LONG | , /STRING | , /UINT | , /UL64 | , /ULONG]
[, TYPE=*value*] [, *Thread pool keywords*])

INT_2D - Computes the double integral of a bivariate function.
Result = INT_2D(*Fxy, AB_Limits, PQ_Limits, Pts*
[, /DOUBLE] [, /ORDER])

INT_3D - Computes the triple integral of a trivariate function.
Result = INT_3D(*Fxyz, AB_Limits, PQ_Limits, UV_Limits, Pts* [, /DOUBLE])

INT_TABULATED - Integrates a tabulated set of data.
Result = INT_TABULATED(*X, F* [, /DOUBLE]
[, /SORT])

INTARR - Creates an integer vector or array.
Result = INTARR(*D_1 [, ...,D_8]* [, /NOZERO])

INTERPOL - Performs linear interpolation on vectors.
For regular grids: *Result* = INTERPOL(V, N
[, /LSQUADRATIC] [, /QUADRATIC] [, /SPLINE])
For irregular grids: *Result* = INTERPOL(V, X, U
[, /LSQUADRATIC] [, /QUADRATIC] [, /SPLINE])

INTERPOLATE - Returns an array of interpolates.
Result = INTERPOLATE(*P, X* [, *Y* [, *Z*]]
[, CUBIC=*value*{-1 to 0}] [, /GRID] [, MISSING=*value*]
[, *Thread pool keywords*])

INTERVAL_VOLUME - Generates a tetrahedral mesh from volumetric data.
INTERVAL_VOLUME, *Data, Value0, Value1, Outverts, Outconn* [, AUXDATA_IN=*array,*
AUXDATA_OUT=*variable*]
[, GEOM_XYZ=*array,* TETRAHEDRA=*array*]
[, PROGRESS_CALLBACK=*string*]
[, PROGESS_METHOD=*string*]
[, PROGRESS_OBJECT=*objref*]
[, PROGRESS_PERCENT=*percent*{0 to 100}]
[, PROGRESS_USERDATA=*value*]

INVERT - Computes the inverse of a square array.
Result = INVERT(*Array* [, *Status*] [, /DOUBLE])

IOCTL - Performs special functions on UNIX files.
Result = IOCTL(*File_Unit* [, *Request, Arg*]
[, /BY_VALUE] [, /MT_OFFLINE] [, /MT_REWIND]
[, MT_SKIP_FILE=[-]*number_of_files*]
[, MT_SKIP_RECORD=[-]*number_of_records*]
[, /MT_WEOF] [, /SUPRESS_ERROR])

IPLOT - Creates an iTool and associated user interface (UI) configured to display and manipulate plot data.
IPLOT, [*X,*] *Y*
or
IPLOT, *X, Y, Z*
ot
IPLOT[, *R*], *Theta*, /POLAR
iTool Common Keywords:
[, BACKGROUND_COLOR=*value*]
[, DIMENSIONS=[*x, y*]]
[, /DISABLE_SPLASH_SCREEN]
[, IDENTIFIER=*variable*] [, LOCATION=[*x, y*]]
[, MACRO_NAMES=*string or string array*]
[, NAME=*string*] [. /NO_SAVEPROMPT]
[, OVERPLOT=*iToolID*] {, STYLE_NAME=*string*]
[, TITLE=*string*] [, VIEW_GRID=[*columns, rows*]]
[, /VIEW_NEXT] [, VIEW_NUMBER=*integer*]
[, VIEW_TITLE=*string*]
iTool Plot Keywords: [, ERRORBAR_COLOR=*RGB vector*] [, ERROR_CAPSIZE=*points*{0.0 to 1.0}]
[, /FILL_BACKGROUND] [, FILL_COLOR=*RGB vector*] [, FILL_LEVEL=*value*] [, RGB_TABLE=*byte array of 256 by 3 or 3 by 256 elements*] [, /SCATTER]
[, SYM_COLOR=*RGB color*]
[, SYM_INCREMENT=*integer*]
[, SYM_INDEX=*integer*] [, SYM_SIZE=*point*{0.0 to 1.0}] [, SYM_THICK=*points*{1.0 to 10.0}]
[, TRANSPARENCY=*percent*{0.0 to 100.0}]
[, /USE_DEFAULT_COLOR] [, /XY_SHADOW]
[, /{X | Y | Z}_ERRORBARS] [, /{X | Y | Z}_LOG]
[, {X | Y | Z}ERROR=*vector* or *array*]
[, /XZ_SHADOW] [, /YZ_SHADOW]
Plot Object Keywords: [, CLIP_PLANES=*array*]
[, COLOR = *RGB vector*] [, /HIDE] [, /HISTOGRAM]
[, LINESTYLE=*integer*] [, MAX_VALUE=*value*]
[, MIN_VALUE=*value*] [, NSUM=*value*] [, /POLAR]
[, THICK=*points*{1.0 to 10.0}]
[, VERT_COLORS=*vector*]

IPLOT - *continued*

Axis Object Keywords: [, {X | Y | Z}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, {X | Y | Z}MAJOR=*integer*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=[*min*, *max*]] [, {X | Y | Z} SUBTICKLEN=*ratio*] [, {X | Y | Z}TEXT_COLOR=*RGB vector*] [, {X | Y | Z}TICKFONT_INDEX= {0 | 1 | 2 | 3 | 4}] [, {X | Y | Z}TICKFONT_SIZE=*integer*] [, {X | Y | Z} TICKFONT_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z} TICKFORMAT=*string or string array*] [, {X | Y | Z} TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT= {0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z} TICKNAME=*string array*] [, {X | Y | Z}TICKUNITS= *string*] [, {X | Y | Z}TICKVALUES=*vector*] [, {X | Y | Z} TITLE=*string*]

ISHFT - Performs integer bit shift.

Result = ISHFT(*P*$_1$, *P*$_2$ [, *Thread pool keywords*])

ISOCONTOUR - Interprets the contouring algorithm found in the IDLgrContour object.

ISOCONTOUR, *Values*, *Outverts*, *Outconn* [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*] [, C_LABEL_INTERVAL=*vector of values*] [, C_LABEL_SHOW=*vector of integers*] [, C_VALUE=*scalar or vector*] [, /DOUBLE] [, /FILL] [, GEOMX=*vector*] [, GEOMY=*vector*] [, GEOMZ=*vector*] [, LEVEL_VALUES=*variable*] [, N_LEVELS=*levels*] [, OUT_LABEL_OFFSETS=*variable*] [, OUT_LABEL_POLYLINES=*variable*] [, OUT_LABEL_STRINGS=*variable*] [, OUTCONN_INDICES=*variable*] [, POLYGONS=*array of polygon descriptions*]

ISOSURFACE - Returns topologically consistent triangles by using oriented tetrahedral decomposition.

ISOSURFACE, *Data*, *Value*, *Outverts*, *Outconn* [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*] [, GEOM_XYZ=*array*, TETRAHEDRA=*array*] [, PROGRESS_CALLBACK=*string*] [, PROGESS_METHOD=*string*] [, PROGRESS_OBJECT=*objref*] [, PROGRESS_PERCENT=*percent*{0 to 100}] [, PROGRESS_USERDATA=*value*]

ISURFACE - Creates an iTool and associated user interface (UI) configured to display and manipulate surface data.

ISURFACE[, *Z*[, *X*, *Y*]]

iTool Common Keywords: [, BACKGROUND_COLOR=*value*] [, DIMENSIONS=[*x*, *y*]] [, /DISABLE_SPLASH_SCREEN] [, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]] [, MACRO_NAMES=*string or string array*] [, NAME=*string*] [, /NO_SAVEPROMPT] [, OVERPLOT=*iToolID*] {, STYLE_NAME=*string*] [, TITLE=*string*] [, VIEW_GRID=[*columns*, *rows*]] [, /VIEW_NEXT] [, VIEW_NUMBER=*integer*] [, VIEW_TITLE=*string*]

iTool Surface Keywords: [, RGB_TABLE=*array of 256 by 3 or 3 by 256 elements*] [, TEXTURE_ALPHA=*2-D array*] [, TEXTURE_BLUE=*2-D array*] [, TEXTURE_GREEN=*2-D array*] [, TEXTURE_IMAGE=*array*] [, TEXTURE_RED=*2-D array*]

Surface Object Keywords: [, BOTTOM=*index or RGB vector*] [, CLIP_PLANES=*array*] [, COLOR=*RGB vector*] [, DEPTH_OFFSET=*value*] [, /DOUBLE] [, /EXTENDED_LEGO] [, /HIDDEN_LINES] [, /HIDE] [, LINESTYLE=*value*] [, SHADING={0 | 1}] [, /SHOW_SKIRT] [, SKIRT=*Z value*] [, STYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, TEXTURE_HIGHRES={0 | 1 | 2}] [, /TEXTURE_INTERP] [, THICK=*points*{1.0 to 10.0}] [, /USE_TRIANGLES] [, VERT_COLORS=*vector or 2-D array*] [, ZERO_OPACITY_SKIP={0 | 1}]

Axis Object Keywords: [, {X | Y | Z}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, {X | Y | Z}MAJOR=*integer*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=[*min*, *max*]] [, {X | Y | Z} SUBTICKLEN=*ratio*] [, {X | Y | Z}TEXT_COLOR=*RGB vector*] [, {X | Y | Z}TICKFONT_INDEX= {0 | 1 | 2 | 3 | 4}] [, {X | Y | Z}TICKFONT_SIZE=*integer*] [, {X | Y | Z} TICKFONT_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z} TICKFORMAT=*string or string array*] [, {X | Y | Z} TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT= {0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z} TICKNAME=*string array*] [, {X | Y | Z}TICKUNITS= *string*] [, {X | Y | Z}TICKVALUES=*vector*] [, {X | Y | Z} TITLE=*string*]

ITCURRENT - Set the current tool in the iTools system.

ITCURRENT, *iToolID*

ITDELETE - Deletes a tool in the iTools system.

ITDELETE[, *iToolID*]

ITGETCURRENT - Gets the identifier of the current iTool.

Result = ITGETCURRENT([, TOOL=*variable*])

ITREGISTER - Registers tool object classes with the iTools system.

ITREGISTER, *Name*, *ItemName* [, /ANNOTATION] [, /DEFAULT] [, /FILE_READER] [, /FILE_WRITER] [, TYPES=*string*] [, /UI_PANEL] [, /UI_SERVICE] [, /USER_INTERFACE] [, /VISUALIZATION]

ITRESET - Resets the iTools session.

ITRESET[, /NO_PROMPT]

ITRESOLVE - Resolves all IDL code within the iTools directory, as well as all other IDL code required for the iTools framework.
ITRESOLVE[, PATH=*string*]

IVOLUME - Creates an iTool and associated user interface (UI) configured to display and manipulate volume data.
IVOLUME[, Vol_0[, Vol_1][, Vol_2, Vol_3]]
iTool Common Keywords:
[, BACKGROUND_COLOR=*value*]
[, DIMENSIONS=[*x*, *y*]]
[, /DISABLE_SPLASH_SCREEN]
[, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]]
[, MACRO_NAMES=*string or string array*]
[, NAME=*string*] [, /NO_SAVEPROMPT]
[, OVERPLOT=*iToolID*] {, STYLE_NAME=*string*]
[, TITLE=*string*] [, VIEW_GRID=[*columns*, *rows*]]
[, /VIEW_NEXT] [, VIEW_NUMBER=*integer*]
[, VIEW_TITLE=*string*]
iTool Volume Keywords: [, /AUTO_RENDER]
[, RENDER_EXTENTS={0 | 1 | 2}]
[, RENDER_QUALITY={1 | 2}]
[, SUBVOLUME=[*xmin*, *ymin*, *zmin*, *xmax*, *ymax*, *zmax*]]
[, VOLUME_DIMENSIONS=[*width*, *height*, *depth*]]
[, VOLUME_LOCATION=[*x*, *y*, *z*]]
Volume Object Keywords: [, AMBIENT=*RGB vector*]
[, BOUNDS=[*xmin*, *ymin*, *zmin*, *xmax*, *ymax*, *zmax*]]
[, CLIP_PLANES=*array*]
[, COMPOSITE_FUNCTION={0 | 1 | 2 | 3}]
[, DEPTH_CUE=[*zbright*, *zdim*]] [, /HIDE]
[, HINTS={0 | 1 | 2 | 3}] [, /INTERPOLATE]
[, /LIGHTING_MODEL]
[, OPACITY_TABLE0=*byte array of 256 elements*]
[, OPACITY_TABLE1=*byte array of 256 elements*]
[, RENDER_STEP=[*x*, *y*, *z*]]
[, RGB_TABLE0=*byte array of 256 by 3 or 3 by 256 elements*] [, RGB_TABLE1=*byte array of 256 by 3 or 3 by 256 elements*] [, /TWO_SIDED] [, /ZBUFFER]
[, ZERO_OPACITY_SKIP={0 | 1}]
Axis Object Keywords: [, {X | Y | Z}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, {X | Y | Z}MAJOR=*integer*]
[, {X | Y | Z}MINOR=*integer*]
[, {X | Y | Z}RANGE=[*min*, *max*]]
[, {X | Y | Z}SUBTICKLEN=*ratio*]
[, {X | Y | Z}TEXT_COLOR=*RGB vector*]
[, {X | Y | Z}TICKFONT_INDEX= {0 | 1 | 2 | 3 | 4}]
[, {X | Y | Z}TICKFONT_SIZE=*integer*] [, {X | Y | Z} TICKFONT_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z} TICKFORMAT=*string or string array*] [, {X | Y | Z} TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT= {0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z} TICKNAME=*string array*] [, {X | Y | Z}TICKUNITS= *string*] [, {X | Y | Z}TICKVALUES=*vector*]
[, {X | Y | Z} TITLE=*string*]

J

JOURNAL - Logs IDL commands to a file.IDL.
JOURNAL [, *Arg*]

JULDAY - Returns Julian Day Number for given month, day, and year.
Result = JULDAY(*Month*,*Day*,*Year*,*Hour*,*Minute*,*Second*)

K

KEYWORD_SET - Returns True if *Expression* is defined and non-zero or an array.
Result = KEYWORD_SET(*Expression*)

KRIG2D - Interpolates set of points using kriging.
Result = KRIG2D(*Z* [, *X*, *Y*] [, EXPONENTIAL=*vector*]
[, SPHERICAL=*vector*] [, /REGULAR]
[, XGRID=*[xstart, xspacing]*] [, XVALUES=*array*]
[, YGRID=*[ystart, yspacing]*] [, YVALUES=*array*]
[, GS=*[xspacing, yspacing]*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*] [, NX=*value*] [, NY=*value*])

KURTOSIS - Computes statistical kurtosis of *n*-element vector.
Result = KURTOSIS(*X* [, /DOUBLE] [, /NAN])

KW_TEST - Performs Kruskal-Wallis H-test.
Result = KW_TEST(*X* [, DF=*variable*]
[, MISSING=*nonzero_value*])

L

L64INDGEN - Returns a 64-bit integer array with each element set to its subscript.
Result = L64INDGEN(D_1 *[, ...,*D_8*]* [, *Thread pool keywords*])

LABEL_DATE - Labels axes with dates. Use with [XYZ]TICKFORMAT keyword.
Result = LABEL_DATE
([DATE_FORMAT=*string/string array*]
[, AM_PM=2-element vector of strings]
[, DAYS_OF_WEEK=7-element vector of strings]
[, MONTHS=12-element vector of strings]
[, OFFSET=*value*] [, /ROUND_UP])
and then,
PLOT, *x*, *y*, XTICKFORMAT = 'LABEL_DATE'

LABEL_REGION - Labels regions (blobs) of a bi-level image.
Result = LABEL_REGION(*Data* [, /ALL_NEIGHBORS]
[, /ULONG])

LADFIT - Fits paired data using least absolute deviation method.
Result = LADFIT(*X*, *Y* [, ABSDEV=*variable*]
[, /DOUBLE])

LAGUERRE - Returns value of the associated Laguerre polynomial.

Result = LAGUERRE(*X*, *N* [, *K*]
[, COEFFICIENTS=*variable*] [, /DOUBLE])

LA_CHOLDC - Computes the Cholesky factorization of an *n*-by-*n* symmetric (or Hermitian) positive-definite array.

LA_CHOLDC, *Array* [, /DOUBLE] [, STATUS=*variable*]
[, /UPPER]

LA_CHOLMPROVE - Uses Cholesky factorization to improve the solution to a system of linear equations.

Result = LA_CHOLMPROVE(*Array, Achol, B, X*
[, BACKWARD_ERROR=*variable*] [, /DOUBLE]
[, FORWARD_ERROR=*variable*] [, /UPPER])

LA_CHOLSOL - Used in conjunction with the LA_CHOLDC procedure to solve a set of linear equations.

Result = LA_CHOLSOL(*A*, *B* [, /DOUBLE] [, /UPPER])

LA_DETERM - Uses LU decomposition to compute the determinant of a square array.

Result = LA_DETERM(*A* [, /CHECK] [, /DOUBLE]
[, ZERO=*value*])

LA_EIGENPROBLEM - Uses the QR algorithm to compute all eigenvalues and eigenvectors of an array.

Result = LA_EIGENPROBLEM(*A* [, *B*]
[, ALPHA=*variable*] [, BALANCE=*value*]
[, BETA=*variable*] [, /DOUBLE]
[, EIGENVECTORS=*variable*]
[, LEFT_EIGENVECTORS=*variable*]
[, NORM_BALANCE=*variable*]
[, PERMUTE_RESULT=*variable*]
[, SCALE_RESULT=*variable*]
[, RCOND_VALUE=*variable*]
[, RCOND_VECTOR=*variable*] [, STATUS=*variable*])

LA_EIGENQL - Computes eigenvalues and eigenvectors of an array.

Result = LA_EIGENQL(*A* [, B] [, /DOUBLE]
[, EIGENVECTORS=*variable*] [, FAILED=*variable*]
[, GENERALIZED=*value*] [, METHOD=*value*]
[, RANGE=*vector*] [, SEARCH_RANGE=*vector*]
[, STATUS=*variable*] [, TOLERANCE=*value*])

LA_EIGENVEC - Uses the QR algorithm to compute all or some of the eigenvectors of an array.

Result = LA_EIGENVEC(*T, QZ* [, BALANCE=*value*]
[, /DOUBLE] [, EIGENINDEX=*variable*]
[, LEFT_EIGENVECTORS=*variable*]
[, PERMUTE_RESULT=[*ilo*, *ihi*]]
[, SCALE_RESULT=*vector*]
[, RCOND_VALUE=*variable*]
[, RCOND_VECTOR=*variable*] [, SELECT=*vector*])

LA_ELMHES - Reduces a real nonsymmetric or complex non-Hermitian array to upper Hessenberg form *H*.

Result = LA_ELMHES(*Array* [, Q] [, BALANCE=*value*]
[, /DOUBLE] [, NORM_BALANCE=*variable*]
[, PERMUTE_RESULT=*variable*]
[, SCALE_RESULT=*variable*])

LA_GM_LINEAR_MODEL - Used to solve a general Gauss-Markov linear model problem.

Result = LA_GM_LINEAR_MODEL(*A, B, D, Y*
[, /DOUBLE])

LA_HQR - Uses the multishift QR algorithm to compute all eigenvalues of an *n*-by-*n* upper Hessenberg array.

Result = LA_HQR(*H* [, Q] [, /DOUBLE]
[, PERMUTE_RESULT=[*ilo*, *ihi*]]
[, STATUS=*variable*])

LA_INVERT - Uses LU decomposition to compute the inverse of a square array.

Result = LA_INVERT(*A* [, /DOUBLE]
[, STATUS=*variable*])

LA_LEAST_SQUARE_EQUALITY - Used to solve the linear least-squares problem.

Result = LA_LEAST_SQUARE_EQUALITY(*A*, *B*, *C*, *D*
[, /DOUBLE] [, RESIDUAL=*variable*])

LA_LEAST_SQUARES - Used to solve the linear least-squares problem.

Result = LA_LEAST_SQUARES(*A*, *B* [, /DOUBLE]
[, METHOD=*value*] [, RANK=*variable*]
[, RCONDITION=*value*] [, RESIDUAL=*variable*]
[, STATUS=*variable*])

LA_LINEAR_EQUATION - Uses LU decomposition to solve a system of linear equations.

Result = LA_LINEAR_EQUATION(*Array, B*
[, BACKWARD_ERROR=*variable*] [, /DOUBLE]
[, FORWARD_ERROR=*variable*]
[, STATUS=*variable*])

LA_LUDC - Computes the LU decomposition of an *n*-column by *m*-row array.

LA_LUDC, *Array, Index* [, /DOUBLE]
[, STATUS=*variable*]

LA_LUMPROVE - Uses LU decomposition to improve the solution to a system of linear equations.

Result = LA_LUMPROVE(*Array, Aludc, Index, B, X*
[, BACKWARD_ERROR=*variable*] [, /DOUBLE]
[, FORWARD_ERROR=*variable*])

LA_LUSOL - Used in conjunction with the LA_LUDC procedure to solve a set of *n* linear equations in *n* unknowns, *AX* = *B*.

Result = LA_LUSOL(*A, Index, B* [, /DOUBLE])

LA_SVD - procedure computes the singular value decomposition of an *n*-columns by *m*-row array.

LA_SVD, *Array, W, U, V* [, /DOUBLE]
[, /DIVIDE_CONQUER] [, STATUS=*variable*]

LA_TRIDC - Computes the LU decomposition of a tridiagonal array as array.

LA_TRIDC, *AL, A, AU, U2, Index* [, /DOUBLE]
[, STATUS=*variable*]

LA_TRIMPROVE - Improves the solution to a system of linear equations with a tridiagonal array.
Result = LA_TRIMPROVE(*AL*, *A*, *AU*, *DAL*, *DA*, *DAU*, *DU2*, *Index*, *B*, *X* [, BACKWARD_ERROR=*variable*] [, /DOUBLE] [, FORWARD_ERROR=*variable*])

LA_TRIQL - Uses the QL and QR variants of the implicitly-shifted QR algorithm to compute the eigenvalues and eigenvectors of a symmetric tridiagonal array.
LA_TRIQL, D, E [, A] [, /DOUBLE] [, STATUS=*variable*]

LA_TRIRED - Reduces a real symmetric or complex Hermitian array to real tridiagonal form *T*.
LA_TRIRED, *Array*, *D*, *E* [, /DOUBLE] [, /UPPER]

LA_TRISOL - Used in conjunction with the LA_TRIDC procedure to solve a set of linear equations.
Result = LA_TRISOL(*AL*, *A*, *AU*, *U2*, *Index*, *B* [, /DOUBLE])

LEEFILT - Performs the Lee filter algorithm on an image array.
Result = LEEFILT(*A* [, *N* [, *Sig*]] [, /DOUBLE] [, /EXACT])

LEGENDRE - Returns value of the associated Legendre polynomial.
Result = LEGENDRE(*X*, *L* [, *M*] [, /DOUBLE])

LINBCG - Solves a set of sparse linear equations using the iterative biconjugate gradient method.
Result = LINBCG(*A*, *B*, *X* [, /DOUBLE] [, ITOL={4 | 5 | 6 | 7}] [, TOL=*value*] [, ITER=*variable*] [, ITMAX=*value*])

LINDGEN - Returns a longword integer array with each element set to its subscript.
Result = LINDGEN(D_1 *[, ...,*D_8*]* [, *Thread pool keywords*])

LINFIT - Fits by minimizing the Chi-square error statistic.
Result = LINFIT(*X*, *Y* [, CHISQ=*variable*] [, COVAR=*variable*] [, /DOUBLE] [, MEASURE_ERRORS=*vector*] [, PROB=*variable*] [, SIGMA=*variable*] [, YFIT=*variable*])

LINKIMAGE - Merges routines written in other languages with IDL at run-time.
LINKIMAGE, *Name*, *Image* [, *Type* [, *Entry*]] [, /DEVICE] [, /FUNCT] [, /KEYWORDS] [, MAX_ARGS=*value*] [, MIN_ARGS=*value*]

LL_ARC_DISTANCE - Returns the longitude and latitude of a point given arc distance and azimuth.
Result = LL_ARC_DISTANCE(*Lon_lat0*, *Arc_Dist*, *Az* [, /DEGREES])

LMFIT - Does a non-linear least squares fit.
Result = LMFIT(*X*, *Y*, *A* [, ALPHA=*variable*] [, CHISQ=*variable*] [, CONVERGENCE=*variable*] [, COVAR=*variable*] [, /DOUBLE] [, FITA=*vector*] [, FUNCTION_NAME=*string*] [, ITER=*variable*] [, ITMAX=*value*] [, ITMIN=*value*] [, MEASURE_ERRORS=*vector*] [, SIGMA=*variable*] [, TOL=*value*])

LMGR - Determines the type of license used by the current IDL session.
Result = LMGR([, /CLIENTSERVER | , /DEMO | , /EMBEDDED | , /RUNTIME | , /STUDENT | , /TRIAL | , /VM] [, EXPIRE_DATE=*variable*] [, /FORCE_DEMO] [, INSTALL_NUM=*variable*] [, LMHOSTID=*variable*] [, SITE_NOTICE=*variable*])

LNGAMMA - Returns logarithm of the gamma function of *Z*.
Result = LNGAMMA(*Z* [, *Thread pool keywords*])

LNP_TEST - Computes the Lomb Normalized Periodogram.
Result = LNP_TEST(*X*, *Y* [, /DOUBLE] [, HIFAC=*scale_factor*] [, JMAX=*variable*] [, OFAC=*value*] [, WK1=*variable*] [, WK2=*variable*])

LOADCT - Loads one of the predefined IDL color tables.
LOADCT [, *Table*] [, BOTTOM=*value*] [, FILE=*string*] [, GET_NAMES=*variable*] [, NCOLORS=*value*] [, /SILENT]

LOCALE_GET - Returns the current locale of the operating platform.
Result = LOCALE_GET( )

LOGICAL_AND - Performs a logical AND operation on its arguments.
Result = LOGICAL_AND(*Arg1*, *Arg2*)

LOGICAL_OR - Performs a logical OR operation on its arguments.
Result = LOGICAL_OR(*Arg1*, *Arg2*)

LOGICAL_TRUE - Determines whether its arguments are non-zero (or non-NULL).
Result = LOGICAL_TRUE(*Arg*)

LON64ARR - Returns a 64-bit integer vector or array.
Result = LON64ARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

LONARR - Returns a longword integer vector or array.
Result = LONARR(D_1 *[, ...,*D_8*]* [, /NOZERO])

LONG - Converts argument to longword integer type.
Result = LONG(*Expression [, Offset [,* D_1 *[, ...,*D_8*]]]* [, *Thread pool keywords*])

LONG64 - Converts argument to 64-bit integer type.
Result = LONG64(*Expression [, Offset [,* D_1 *[, ...,*D_8*]]]* [, *Thread pool keywords*])

LSODE - Advances a solution to a system of ordinary differential equations one time-step H.
Result = LSODE(Y, X, H, Derivs[, Status] [, ATOL=*value*] [, /QUIET] [, RTOL=*value*])

LU_COMPLEX - Solves complex linear system using LU decomposition.
Result = LU_COMPLEX(*A*, *B* [, /DOUBLE] [, /INVERSE] [, /SPARSE])

LUDC - Replaces array with the LU decomposition.
LUDC, *A*, *Index* [, /COLUMN] [, /DOUBLE] [, INTERCHANGES=*variable*]

LUMPROVE - Uses LU decomposition to iteratively improve an approximate solution.
Result = LUMPROVE(*A, Alud, Index, B, X* [, /COLUMN] [, /DOUBLE])

LUSOL - Solves a set of linear equations. Use with LUDC.
Result = LUSOL(*A, Index, B* [, /COLUMN] [, /DOUBLE])

M

M_CORRELATE - Computes multiple correlation coefficient.
Result = M_CORRELATE(*X, Y* [, /DOUBLE])

MACHAR - Determines and returns machine-specific parameters affecting floating-point arithmetic.
Result = MACHAR([, /DOUBLE])

MAKE_ARRAY - Returns an array of the specified type, dimensions, and initialization.
Result = MAKE_ARRAY (*[D_1 [, ...,D_8]]* [, /BYTE | , /COMPLEX | , /DCOMPLEX | , /DOUBLE | , /FLOAT | , /INT | , /L64 | , /LONG | , /OBJ, | , /PTR | , /STRING | , /UINT | , /UL64 | , /ULONG] [, DIMENSION=*vector*] [, /INDEX] [, /NOZERO] [, SIZE=*vector*] [, TYPE=*type_code*] [, VALUE=*value*] [, *Thread pool keywords*])

MAKE_DLL - Builds a shareable library suitable for use with IDL's dynamic linking.
MAKE_DLL, *InputFiles* [, *OutputFile*], *ExportedRoutineNames* [, CC=*string*] [, COMPILE_DIRECTORY=*path*] [, DLL_PATH=*variable*] [, EXPORTED_DATA=*string*] [, EXTRA_CFLAGS=*string*] [, EXTRA_LFLAGS=*string*] [, INPUT_DIRECTORY=*path*] [, LD=*string*] [, /NOCLEANUP] [, OUTPUT_DIRECTORY=*path*] [, /REUSE_EXISTING] [, /SHOW_ALL_OUTPUT] [, /VERBOSE]

MAP_2POINTS - Returns distance, azimuth, and path relating to the great circle or rhumb line connecting two points on a sphere.
Result = MAP_2POINTS(*lon0, lat0, lon1, lat1* [, DPATH=*value* | , /METERS | , /MILES | , NPATH=*integer*{2 or greater} | , /PARAMETERS | , RADIUS=*value*] [, /RADIANS] [, /RHUMB])

MAP_CONTINENTS - Draws continental boundaries, filled continents, political boundaries, coastlines, and/or rivers, over an existing map projection established by MAP_SET.
MAP_CONTINENTS [, /COASTS] [, COLOR=*index*] [, /CONTINENTS] [, /COUNTRIES] [,FILL_CONTINENTS={1 | 2}] [, ORIENTATION=*value*]] [, /HIRES] [, LIMIT=*vector*] [, MAP_STRUCTURE=*structure*] [, MLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, MLINETHICK=*value*] [, /RIVERS] [, SPACING=*centimeters*] [, /USA]
Graphics Keywords: [, /T3D][, ZVALUE=*value*{0 to 1}]

MAP_GRID - Draws parallels and meridians over a map projection.
MAP_GRID [, /BOX_AXES | [, CLIP_TEXT=0] [, LATALIGN=*value*{0.0 to 1.0}] [, LONALIGN=*value*{0.0 to 1.0}] [, LATLAB=*longitude*] [, LONLAB=*latitude*] [, ORIENTATION=*clockwise_degrees_from_horiz*]] [, CHARSIZE=*value*] [, COLOR=*index*] [, /FILL_HORIZON] [, GLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, GLINETHICK=*value*] [, /HORIZON] [, INCREMENT=*value*] [, LABEL=*n*{label_every_*n*th_gridline}] [, LATDEL=*degrees*] [, LATNAMES=*array*, LATS=*vector*] [, LONDEL=*degrees*] [, LONNAMES=*array*, LONS=*vector*] [, MAP_STRUCTURE=*structure*] [, /NO_GRID]
Graphics Keywords: [, /T3D] [, ZVALUE=*value*{0 to 1}]

MAP_IMAGE - Returns an image warped to fit the current map projection. (Use when map data is larger than the display).
Result = MAP_IMAGE(*Image* [, *Startx*, *Starty* [, *Xsize*, *Ysize*]] [, LATMIN=*degrees*{-90 to 90}] [, LATMAX=*degrees*{-90 to 90}] [, LONMIN=*degrees*{-180 to 180}] [, LONMAX=*degrees*{-180 to 180}] [, /BILINEAR] [, COMPRESS=*value*] [, SCALE=*value*] [, MAP_STRUCTURE=*structure*] [, MASK=*variable*] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, MISSING=*value*])

MAP_PATCH - Returns an image warped to fit the current map projection. (Use when map data is smaller than the display).
Result = MAP_PATCH(*Image_Orig* [, *Lons*, *Lats*] [, LAT0=*value*] [, LAT1=*value*] [, LON0=*value*] [, LON1=*value*] [, MAX_VALUE=*value*] [, MISSING=*value*] [, /TRIANGULATE] [, XSIZE=*variable*] [, XSTART=*variable*] [, YSIZE=*variable*] [, YSTART=*variable*])

MAP_PROJ_FORWARD - Transforms map coordinates from longitude/latitude to Cartesian (X, Y) coordinates.
Result = MAP_PROJ_FORWARD(*Longitude* [, *Latitude*] [, CONNECTIVITY=*vector*] [, /FILL] [, MAP_STRUCTURE=*value*] [, POLYGONS=*variable*] [, POLYLINES=*variable*] [, /RADIANS] [, *Thread pool keywords*])

MAP_PROJ_IMAGE - Warps an image from geographic coordinates to a specified map projection.
Result = MAP_PROJ_IMAGE(*Image* [, *Range*] [, /BILINEAR] [, DIMENSIONS=*vector*] [, IMAGE_STRUCTURE=*structure*] [, MAP_STRUCTURE=*structure*] [, MASK=*variable*] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, MISSING=*value*] [, UVRANGE=*variable*] [, XINDEX=*variable*] [, YINDEX=*variable*])

MAP_PROJ_INFO - Returns information about current map and/or the available projections.

MAP_PROJ_INFO [, *iproj*] [, AZIMUTHAL=*variable*] [, CIRCLE=*variable*] [, CYLINDRICAL=*variable*] [, /CURRENT] [, LL_LIMITS=*variable*] [, NAME=*variable*] [, PROJ_NAMES=*variable*] [, UV_LIMITS=*variable*] [, UV_RANGE=*variable*]

MAP_PROJ_INIT - Initializes a mapping projection.

Result = MAP_PROJ_INIT(*Projection* [, DATUM=*value*] [, /GCTP] [, LIMIT=*vector*] [, /RADIANS] [, /RELAXED])

Keywords—Projection Parameters:

[, CENTER_AZIMUTH=*value*] [, CENTER_LATITUDE=*value*] [, CENTER_LONGITUDE=*value*] [, FALSE_EASTING=*value*] [, FALSE_NORTHING=*value*] [, HEIGHT=*value*] [, HOM_AZIM_LONGITUDE=*value*] [, HOM_AZIM_ANGLE=*value*] [, HOM_LATITUDE1=*value*] [, HOM_LATITUDE2=*value*] [, HOM_LONGITUDE1=*value*] [, HOM_LONGITUDE2=*value*] [, IS_ZONES=*value*] [, IS_JUSTIFY=*value*] [, MERCATOR_SCALE=*value*] [, OEA_ANGLE=*value*] [, OEA_SHAPEM=*value*] [, OEA_SHAPEN=*value*] [, ROTATION=*value*] [, SEMIMAJOR_AXIS=*value*] [, SEMIMINORAXIS=*value*] [, SOM_INCLINATION=*value*] [, SOM_LONGITUDE=*value*] [, SOM_PERIOD=*value*] [, SOM_RATIO=*value*] [, SOM_FLAG=*value*] [, SOM_LANDSAT_NUMBER=*value*] [, SOM_LANDSAT_PATH=*value*] [, SPHERE_RADIUS=*value*] [, STANDARD_PARALLEL=*value*] [, STANDARD_PAR1=*value*] [, STANDARD_PAR2=*value*] [, SAT_TILT=*value*] [, TRUE_SCALE_LATITUDE=*value*] [, ZONE=*value*]

MAP_PROJ_INVERSE - Transforms map coordinates from Cartesian (X, Y) coordinates to longitude/latitude.

Result = MAP_PROJ_INVERSE (*X* [, *Y*] [, MAP_STRUCTURE=*value*] [, /RADIANS] [, *Thread pool keywords*])

MAP_SET - Establishes map projection type and limits.

MAP_SET [, *P0lat, P0lon, Rot*]

Keywords—Projection Types: [[, /AITOFF | , /ALBERS | , /AZIMUTHAL | , /CONIC | , /CYLINDRICAL | , /GNOMIC | , /GOODESHOMOLOSINE | , /HAMMER | , /LAMBERT | , /MERCATOR | , /MILLER_CYLINDRICAL | , /MOLLWEIDE | , /ORTHOGRAPHIC | , /ROBINSON | , /SATELLITE | , /SINUSOIDAL | , /STEREOGRAPHIC | , /TRANSVERSE_MERCATOR] | NAME=*string*]]

Keywords—Map Characteristics: [, /ADVANCE] [, CHARSIZE=*value*] [, /CLIP] [, COLOR=*index*] [, /CONTINENTS [, CON_COLOR=*index*] [, /HIRES]] [, E_CONTINENTS=*structure*] [, E_GRID=*structure*] [, E_HORIZON=*structure*] [, GLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, GLINETHICK=*value*] [, /GRID] [, /HORIZON] [, LABEL=*n*{label every *n*th gridline}] [, LATALIGN=*value*{0.0 to 1.0}] [, LATDEL=*degrees*] [, LATLAB=*longitude*] [, LONALIGN=*value*{0.0 to 1.0}] [, LONDEL=*degrees*] [, LONLAB=*latitude*] [, MLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, MLINETHICK=*value*] [, /NOBORDER] [, /NOERASE] [, REVERSE={0 | 1 | 2 | 3}] [, TITLE=*string*] [, /USA] [, XMARGIN=*value*] [, YMARGIN=*value*]

Keywords—Projection Parameters: [, CENTRAL_AZIMUTH=*degrees_east_of_north*] [, ELLIPSOID=*array*] [, /ISOTROPIC] [, LIMIT=*vector*] [, SAT_P=*vector*] [, SCALE=*value*] [, STANDARD_PARALLELS=*array*]

Graphics Keywords: [, POSITION=$[X_0, Y_0, X_1, Y_1]$] [, /T3D] [, ZVALUE=*value*{0 to 1}]

MATRIX_MULTIPLY - Calculates the IDL matrix-multiply operator (#) of two (possibly transposed) arrays.

Result = MATRIX_MULTIPLY(A, B [, /ATRANSPOSE] [, /BTRANSPOSE] [, *Thread pool keywords*])

MATRIX_POWER - Computes the product of a matrix with itself.

Result = MATRIX_POWER(*Array, N* [, /DOUBLE] [, STATUS=*value*])

MAX - Returns the value of the largest element of *Array.*

Result = MAX(*Array* [, *Max_Subscript*] [, /ABSOLUTE] [, DIMENSION=*value*] [, MIN=*variable*] [, /NAN] [, SUBSCRIPT_MIN=*variable*] [, *Thread pool keywords*])

MD_TEST - Performs the Median Delta test.

Result = MD_TEST(*X* [, ABOVE=*variable*] [, BELOW=*variable*] [, MDC=*variable*])

MEAN - Computes the mean of a numeric vector.

Result = MEAN(*X* [, /DOUBLE] [, /NAN])

MEANABSDEV - Computes the mean absolute deviation of a vector.

Result = MEANABSDEV(*X* [, /DOUBLE] [, /MEDIAN] [, /NAN])

MEDIAN - Returns the median value of *Array* or applies a median filter.

Result = MEDIAN(*Array* [, *Width*] [, /DOUBLE] [, DIMENSION=*value*] [, /EVEN])

MEMORY - Returns a vector containing information on the amount of dynamic memory currently in use by the IDL session.

Result = MEMORY([, /CURRENT | , /HIGHWATER | , /NUM_ALLOC | , /NUM_FREE | , /STRUCTURE] [/L64])

MESH_CLIP - Clips a polygonal mesh to an arbitrary plane in space and returns a polygonal mesh of the remaining portion.

Result = MESH_CLIP (*Plane*, *Vertsin*, *Connin*, *Vertsout*, *Connout* [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*][, CUT_VERTS=*variable*])

MESH_DECIMATE - Reduces the density of geometry while preserving as much of the original data as possible.

Result = MESH_DECIMATE (*Verts*, *Conn*, *Connout* [, VERTICES=*variable*] [, PERCENT_VERTICES=*percent* | , PERCENT_POLYGONS=*percent*] [, PROGRESS_CALLBACK=*string*] [, PROGESS_METHOD=*string*] [, PROGRESS_OBJECT=*objref*] [, PROGRESS_PERCENT=*percent*{0 to 100}] [, PROGRESS_USERDATA=*value*])

MESH_ISSOLID - Computes various mesh properties and enables IDL to determine if a mesh encloses space (is a solid).

Result = MESH_ISSOLID (*Conn*)

MESH_MERGE - Merges two polygonal meshes.

Result = MESH_MERGE (*Verts*, *Conn*, *Verts1*, *Conn1* [, /COMBINE_VERTICES] [, TOLERANCE=*value*])

MESH_NUMTRIANGLES - Computes the number of triangles in a polygonal mesh.

Result = MESH_NUMTRIANGLES (*Conn*)

MESH_OBJ - Generates a polygon mesh for various simple objects.

MESH_OBJ, *Type*, *Vertex_List*, *Polygon_List*, *Array1* [, *Array2*] [, /CLOSED] [, /DEGREES] [, P1=*value*] [, P2=*value*] [, P3=*value*] [, P4=*value*] [, P5=*value*]

MESH_SMOOTH - Performs spatial smoothing on a polygon mesh.

Result = MESH_SMOOTH (*Verts*, *Conn* [, ITERATIONS=*value*] [, FIXED_VERTICES=*array*] [, /FIXED_EDGE_VERTICES] [, LAMBDA=*value*])

MESH_SURFACEAREA - Computes various mesh properties to determine the mesh surface area, including integration of other properties interpolated on the surface of the mesh.

Result = MESH_SURFACEAREA (*Verts*, *Conn* [, AUXDATA=*array*] [, MOMENT=*variable*])

MESH_VALIDATE - Checks for NaN values in vertices, removes unused vertices, and combines close vertices.

Result = MESH_VALIDATE (*Verts*, *Conn* [, /REMOVE_NAN] [, /PACK_VERTICES] [, /COMBINE_VERTICES] [, TOLERANCE=*value*])

MESH_VOLUME - Computes the volume that the mesh encloses.

Result = MESH_VOLUME (*Verts*, *Conn* [, /SIGNED])

MESSAGE - Issues error and informational messages.

MESSAGE, [[*Text*] | [Arg_1, ... Arg_N] | [, /REISSUE_LAST]] [, BLOCK=*BlockName*] [, /CONTINUE] [, /INFORMATIONAL] [, /IOERROR] [, LEVEL=*CallLevel*] [, NAME=*ErrorName*] [, /NONAME] [, /NOPREFIX] [, /NOPRINT] [, /RESET]

MIN - Returns the value of the smallest element of an array.

Result = MIN(*Array* [, *Min_Subscript*] [, /ABSOLUTE] [, DIMENSION=*value*] [, MAX=*variable*] [, /NAN] [, SUBSCRIPT_MAX] [, *Thread pool keywords*])

MIN_CURVE_SURF - Interpolates over either a plane or a sphere with a minimum curvature surface or a thin-plate-spline surface.

Result = MIN_CURVE_SURF(*Z* [, *X*, *Y*] [, /DOUBLE] [, /TPS] [, /REGULAR] [, /SPHERE [, /CONST]] [, XGRID=*[xstart, xspacing]* | , XVALUES=*array*] [, YGRID=*[ystart, yspacing]* | , YVALUES=*array*] [, GS=*[xspace,yspace]*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*] [, NX=*value*] [, NY=*value*] [, XOUT=*vector*] [, YOUT=*vector*] [, XPOUT=*array*, YPOUT=*array*])

MK_HTML_HELP - Converts text documentation headers to HTML files.

MK_HTML_HELP, *Sources*, *Filename* [, /STRICT] [, TITLE=*string*] [, /VERBOSE]

MODIFYCT - Saves modified color tables in the IDL color table file.

MODIFYCT, *Itab*, *Name*, *R*, *G*, *B* [, FILE=*filename*]

MOMENT - Computes mean, variance, skewness, and kurtosis.

Result = MOMENT(*X* [, /DOUBLE] [, MDEV=*variable*] [, /NAN] [, SDEV=*variable*])

MORPH_CLOSE - Applies closing operator to binary or grayscale image.

Result = MORPH_CLOSE (*Image*, *Structure* [, /GRAY] [, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG] [, VALUES=*array*])

MORPH_DISTANCE - Estimates *N*-dimensional distance maps, which contain for each foreground pixel the distance to the nearest background pixel, using a given norm.

Result = MORPH_DISTANCE (*Data* [, /BACKGROUND] [, NEIGHBOR_SAMPLING={1 | 2 | 3 }] [, /NO_COPY])

MORPH_GRADIENT - Applies the morphological gradient operator to a grayscale image.

Result = MORPH_GRADIENT (*Image*, *Structure* [, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG] [, VALUES=*array*])

MORPH_HITORMISS - Applies the hit-or-miss operator to a binary image.

Result = MORPH_HITORMISS (*Image*, *HitStructure*, *MissStructure*)

MORPH_OPEN - Applies the opening operator to a binary or grayscale image.

Result = MORPH_OPEN (*Image*, *Structure* [, /GRAY] [, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG] [, VALUES=*array*])

MORPH_THIN - Performs a thinning operation on binary images.

Result = MORPH_THIN (*Image*, *HitStructure*, *MissStructure*)

MORPH_TOPHAT - Applies top-hat operator to a grayscale image.
Result = MORPH_TOPHAT (*Image, Structure*
[, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG]
[, VALUES=*array*])

MPEG_CLOSE - Closes an MPEG sequence.
MPEG_CLOSE, *mpegID*

MPEG_OPEN - Opens an MPEG sequence.
mpegID = MPEG_OPEN(*Dimensions*[, BITRATE=*value*]
[, FILENAME=*string*] [, IFRAME_GAP=*integer value*]
[, MOTION_VEC_LENGTH={1 | 2 | 3}]
[QUALITY=*value*{0 to 100}])

MPEG_PUT - Inserts an image array into an MPEG sequence
MPEG_PUT, *mpegID* [, /COLOR]
[, FRAME=*frame_number*] [, IMAGE=*array* | ,
WINDOW=*index*] [, /ORDER]

MPEG_SAVE - Encodes and saves an open MPEG sequence.
MPEG_SAVE, *mpegID* [, FILENAME=*string*]

MULTI - Replicates current color table to enhance contrast.
MULTI, *N*

N

N_ELEMENTS - Returns the number of elements contained in an expression or variable.
Result = N_ELEMENTS(*Expression*)

N_PARAMS - Returns the number of non-keyword parameters used in calling an IDL procedure or function.
Result = N_PARAMS( )

N_TAGS - Returns the number of tags in a structure.
Result = N_TAGS(*Expression* [, /LENGTH])

NCDF_* Routines - See "NetCDF Routines" on page 140.

NEWTON - Solves nonlinear equations using Newton's method.
Result = NEWTON(*X*, *Vecfunc* [, CHECK=*variable*]
[, /DOUBLE] [, ITMAX=*value*] [, STEPMAX=*value*]
[, TOLF=*value*] [, TOLMIN=*value*] [, TOLX=*value*])

NORM - Computes Euclidean norm of vector or Infinity norm of array.
Result = NORM(*A* [, /DOUBLE]
[, LNORM={0 | 1 | 2 | *n*})

O

OBJ_CLASS - Determines the class name of an object.
Result = OBJ_CLASS([*Arg*] [, COUNT=*variable*]
[, /SUPERCLASS{must specify Arg}])

OBJ_DESTROY - Destroys an object reference.
OBJ_DESTROY, *ObjRef* [, Arg_1, …, Arg_n]

OBJ_ISA - Determines inheritance relationship of an object.
Result = OBJ_ISA(*ObjectInstance*, *ClassName*)

OBJ_NEW - Creates an object reference.
Result = OBJ_NEW([*ObjectClassName* [, Arg_1…...Arg_n]])

OBJ_VALID - Verifies validity of object references.
Result = OBJ_VALID([*Arg*] [, /CAST]
[, COUNT=*variable*])

OBJARR - Creates an array of object references.
Result = OBJARR(D_1 *[*, ...,D_8*]* [, /NOZERO])

ON_ERROR - Designates the error recovery method.
ON_ERROR, *N*

ON_IOERROR - Declares I/O error exception handler.
ON_IOERROR, *Label*
. . .
Label: *Statement to perform upon I/O error*

ONLINE_HELP - Invokes online help viewer from programs.
ONLINE_HELP [, *Value*] [, BOOK='*filename*']
[, /FULL_PATH] [, /QUIT]
Windows-Only Keywords: [, /CONTEXT]

OPEN - Opens files for reading, updating, or writing.
OPENR, *Unit, File*
OPENW, *Unit, File*
OPENU, *Unit, File*
Keywords (all platforms): [, /APPEND | , /COMPRESS]
[, BUFSIZE={0 | 1 | *value*>512}] [, /DELETE]
[, ERROR=*variable*] [, /F77_UNFORMATTED]
[, /GET_LUN] [, /MORE] [, /NOEXPAND_PATH]
[, /STDIO] [, /SWAP_ENDIAN]
[, SWAP_IF_BIG_ENDIAN]
[, /SWAP_IF_LITTLE_ENDIAN] [, /VAX_FLOAT]
[, WIDTH=*value*] [, /XDR]
UNIX-Only Keywords: [, /RAWIO]

OPLOT - Plots vector data over a previously-drawn plot.
OPLOT, [*X*,] *Y* [, MAX_VALUE=*value*]
[, MIN_VALUE=*value*] [, NSUM=*value*] [, /POLAR]
[, THICK=*value*]
Graphics Keywords: [, CLIP=*[*X_0*,* Y_0*,* X_1*,* Y_1*]*]
[, COLOR=*value*] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}]
[, /NOCLIP] [, PSYM=*integer*{0 to 10}]
[, SYMSIZE=*value*] [, /T3D] [, ZVALUE=*value*{0 to 1}]

OPLOTERR - Draws error bars over a previously drawn plot.
OPLOTERR, [*X* ,] *Y* , *Err* [, *Psym*]

P

P_CORRELATE - Computes partial correlation coefficient.
Result = P_CORRELATE(*X*, *Y*, *C* [, /DOUBLE])

PARTICLE_TRACE - Traces the path of a massless particle through a vector field.

PARTICLE_TRACE, *Data*, *Seeds*, *Verts*, *Conn* [, *Normals*] [, MAX_ITERATIONS=*value*] [, ANISOTROPY=*array*] [, INTEGRATION={0 | 1}] [, SEED_NORMAL=*vector*] [, TOLERANCE=*value*] [, MAX_STEPSIZE=*value*] [, /UNIFORM]

PATH_CACHE - Used to control IDL's path cache mechanism.

PATH_CACHE [, /CLEAR] [, /ENABLE] [, /REBUILD]

PATH_SEP - Returns the proper file path segment separator character for the current operating system.

Result = PATH_SEP([/PARENT_DIRECTORY] [, /SEARCH_PATH])

PCOMP - Computes principal components/derived variables.

Result = PCOMP(*A* [, COEFFICIENTS=*variable*] [, /COVARIANCE] [, /DOUBLE] [, EIGENVALUES=*variable*] [, NVARIABLES=*value*] [, /STANDARDIZE] [, VARIANCES=*variable*])

PLOT - Plots vector arguments as X versus Y graphs.

PLOT, [*X*,] *Y* [, /ISOTROPIC] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, NSUM=*value*] [, /POLAR] [, THICK=*value*] [, /XLOG] [, /YLOG] [, /YNOZERO]

Graphics Keywords: [, BACKGROUND=*color_index*] [, CHARSIZE=*value*] [, CHARTHICK=*integer*] [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP] [, /NODATA] [, /NOERASE] [, POSITION=*[X_0, Y_0, X_1, Y_1]*] [, PSYM=*integer*{0 to 10}] [, SUBTITLE=*string*] [, SYMSIZE=*value*] [, /T3D] [, THICK=*value*] [, TICKLEN=*value*] [, TITLE=*string*] [, {X | Y | Z}CHARSIZE=*value*] [, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}] [, {X | Y | Z}MARGIN=*[left, right]*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=*[min, max]*] [, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*] [, {X | Y | Z}TICK_GET=*variable*] [, {X | Y | Z}TICKFORMAT=*string*] [, {X | Y | Z}TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT=*scalar*] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string_array*] [, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKUNITS=*string*] [, {X | Y | Z}TICKV=*array*] [, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

PLOT_3DBOX - Plots function of two variables inside 3D box.

PLOT_3DBOX, *X, Y, Z* [, AX=*degrees*] [, AZ=*degrees*] [, GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, PSYM=*integer*{1 to 10}] [, /SOLID_WALLS] [, /XY_PLANE] [, XYSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /XZ_PLANE] [, XZSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /YZ_PLANE] [, YZSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, ZAXIS={1 | 2 | 3 | 4}]

Graphics Keywords: Accepts all graphics keywords accepted by PLOT except for: FONT, PSYM, SYMSIZE, {XYZ}TICK_GET, and ZVALUE.

PLOT_FIELD - Plots a 2D field using arrows.

PLOT_FIELD, *U*, *V* [, ASPECT=*ratio*] [, LENGTH=*value*] [, N=*num_arrows*] [, TITLE=*string*]

PLOTERR - Plots individual data points with error bars.

PLOTERR, [*X* ,] *Y* , *Err* [, TYPE={1 | 2 | 3 | 4}] [, PSYM=*integer*{1 to 10}]

PLOTS - Plots vectors and points.

PLOTS, *X* [, *Y* [, *Z*]] [, /CONTINUE]

Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP] [, PSYM=*integer*{0 to 10}] [, SYMSIZE=*value*] [, /T3D] [, THICK=*value*] [, Z=*value*]

PNT_LINE - Returns the perpendicular distance between a point and a line.

Result = PNT_LINE(*P0*, *L0*, *L1* [, *Pl*] [, /INTERVAL])

POINT_LUN - Sets or gets current position of the file pointer.

POINT_LUN, *Unit*, *Position*

POLAR_CONTOUR - Draws a contour plot from data in polar coordinates.

POLAR_CONTOUR, *Z*, *Theta*, *R* [, C_ANNOTATION=*vector_of_strings*] [, C_CHARSIZE=*value*] [, C_CHARTHICK=*integer*] [, C_COLORS=*vector*] [, C_LINESTYLE=*vector*] [, /FILL | , CELL_FILL [, C_ORIENTATION=*degrees*] [, C_SPACING=*value*]] [, C_THICK=*vector*] [, /CLOSED] [, /IRREGULAR] [, LEVELS=*vector* | NLEVELS=*integer*{1 to 29}] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, /OVERPLOT] [, /PATH_DATA_COORDS | ,TRIANGULATION=*variable*] [, /XLOG] [, /YLOG] [, /ZAXIS] [, SHOW_TRIANGULATION=*color_index*]

POLAR_SURFACE - Interpolates a surface from polar coordinates to rectangular coordinates.

Result = POLAR_SURFACE(*Z*, *R*, *Theta* [, /GRID] [, SPACING=*[xspacing, yspacing]*] [, BOUNDS=*[x_0, y_0, x_1, y_1]*] [, /QUINTIC] [, MISSING=*value*])

POLY - Evaluates polynomial function of a variable.

Result = POLY(*X*, *C*)

POLY_2D - Performs polynomial warping of images.

Result = POLY_2D(*Array, P, Q* [, *Interp* [, Dim_x, Dim_y]] [, CUBIC={-1 to 0}] [, MISSING=*value*] [, *Thread pool keywords*])

POLY_AREA - Returns the area of a polygon given the coordinates of its vertices.

Result = POLY_AREA(*X*, *Y* [, /DOUBLE] [, /SIGNED])

POLY_FIT - Performs a least-square polynomial fit.

Result = POLY_FIT(*X*, *Y*, *Degree* [, CHISQ=*variable*]
[, COVAR=*variable*] [, /DOUBLE]
[, MEASURE_ERRORS=*vector*] [, SIGMA=*variable*]
[, STATUS=*variable*] [, YBAND=*variable*]
[, YERROR=*variable*] [, YFIT=*variable*])

POLYFILL - Fills the interior of a polygon.

POLYFILL, *X* [, *Y* [, *Z*]] [, IMAGE_COORD=*array*]
[, /IMAGE_INTERP] [, /LINE_FILL]
[, PATTERN=*array*] [, SPACING=*centimeters*]
[, TRANSPARENT=*value*]
Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*]
[, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL]
[, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP]
[, ORIENTATION=*ccw_degrees_from_horiz*] [, /T3D]
[, THICK=*value*] [, Z=*value*]

POLYFILLV - Returns subscripts of pixels inside a polygon.

Result = POLYFILLV(*X*, *Y*, S_x, S_y [, *Run_Length*])

POLYSHADE - Creates a shaded surface representation from a set of polygons.

Result = POLYSHADE(*Vertices*, *Polygons*)
or
Result = POLYSHADE(*X*, *Y*, *Z*, *Polygons*)
Keywords: [, /DATA | , /NORMAL]
[, POLY_SHADES=*array*] [, SHADES=*array*] [, /T3D]
[, TOP=*value*] [, XSIZE=*columns*] [, YSIZE=*rows*]

POLYWARP - Performs polynomial spatial warping.

POLYWARP, *Xi*, *Yi*, *Xo*, *Yo*, *Degree*, *Kx*, *Ky* [, /DOUBLE]
[, STATUS=*variable*]

POPD - Removes the top directory on the working directory stack maintained by PUSHD/POPD.

POPD

POWELL - Minimizes a function using the Powell method.

POWELL, *P*, *Xi*, *Ftol*, *Fmin*, *Func* [, /DOUBLE]
[, ITER=*variable*] [, ITMAX=*value*]

PREF_COMMIT - Commits IDL preferences already in the pending state.

PREF_COMMIT [, *PreferenceName*] [, /RESET]
[, /RESIGNAL]

PREF_GET - Returns information about IDL preferences.

Result = PREF_GET([PrefName] [, /NAMES_ALL |
/NUM_PENDING | /STRUCTURE])

PREF_MIGRATE - Imports IDL user preferences from other versions of IDL for use by the currently running version.

PREF_MIGRATE [, /MACRO] [, /PREFERENCE]
[, /STARTUP]

PREF_SET - Sets new values for IDL preferences.

PREF_SET, *PrefName*$_1$, *Value*$_1$ [, ... *PrefName*$_n$, *Value*$_n$]
[, /COMMIT]
or
PREF_SET, *PrefName*$_1$ [, ... *PrefName*$_n$] /DEFAULT
[, /COMMIT]
or
PREF_SET, FILENAME=*file* [, /COMMIT]

PRIMES - Computes the first *K* prime numbers.

Result = PRIMES(*K*)

PRINT/PRINTF - Writes formatted output to screen or file.

PRINT [, *Expr*$_1$, ..., *Expr*$_n$]
PRINTF [, *Unit*, *Expr*$_1$, ..., *Expr*$_n$]
Keywords: [, AM_PM=*[string, string]*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, FORMAT=*value*]
[, MONTHS=*string_array*{12 names}]
[, /STDIO_NON_FINITE]

PRINTD - Prints contents of the directory stack maintained by PUSHD/POPD.

PRINTD

PRO - Defines a procedure.

PRO *Procedure_Name*, *argument*$_1$, ..., *argument*$_n$
. . .
END

PRODUCT - Returns the product of elements within an array.

Result = PRODUCT(*Array* [, *Dimension*]
[, /CUMULATIVE] [, /INTEGER] [, /NAN]
[, /PRESERVE_TYPE])

PROFILE - Extracts a profile from an image.

Result = PROFILE(*Image* [, *XX*, *YY*] [, /NOMARK]
[, XSTART=*value*] [, YSTART=*value*])

PROFILER - Accesses the IDL Code Profiler used to analyze performance of applications.

PROFILER [, *Module*] [, /CLEAR] [, DATA=*variable*]
[, OUTPUT=*variable*] [, /REPORT] [, /RESET]
[, /SYSTEM]

PROFILES - Interactively examines image profiles.

PROFILES, *Image* [, /ORDER] [, SX=*value*] [, SY=*value*]
[, WSIZE=*value*]

PROJECT_VOL - Returns a translucent rendering of a volume projected onto a plane.

Return = PROJECT_VOL(*Vol*, *X_Sample*, *Y_Sample*,
Z_Sample [, /AVG_INTENSITY] [, /CUBIC]
[, DEPTH_Q=*value*] [, OPAQUE=*3D_array*]
[, TRANS=*array*] [, XSIZE=*longword integer*]
[, YSIZE=*longword integer*] [, /Z_BUFFER])

PS_SHOW_FONTS - Displays all the PostScript fonts that IDL knows about.

PS_SHOW_FONTS [, /NOLATIN]

PSAFM - Converts Adobe Font Metrics file to IDL format.
PSAFM, *Input_Filename*, *Output_Filename*

PSEUDO - Creates pseudo-color table based on Lightness, Hue, and Brightness system.
PSEUDO, *Litlo*, *Lithi*, *Satlo*, *Sathi*, *Hue*, *Loops* [, *Colr*]

PTR_FREE - Destroys a pointer.
PTR_FREE, P_1, ... …, P_n

PTR_NEW - Creates a pointer.
Result = PTR_NEW([*InitExpr*] [, /ALLOCATE_HEAP] [, /NO_COPY])

PTR_VALID - Verifies the validity of pointers.
Result = PTR_VALID([*Arg*] [, /CAST] [, COUNT=*variable*])

PTRARR - Creates an array of pointers.
Result = PTRARR(D_1 *[, ...,*D_8*]* [, /ALLOCATE_HEAP | , /NOZERO])

PUSHD - Pushes a directory to top of directory stack maintained by PUSHD/POPD.
PUSHD, *Dir*

Q

QGRID3 - Interpolates the dependent variable values to points in a regularly sampled volume.
Result = QGRID3(*XYZ*, *F*, *Tetrahedra*)
or
Result = QGRID3(*X*, *Y*, *Z*, *F*, *Tetrahedra*)
Keywords: [, DELTA=*array*] [, DIMENSION=*array*] [, MISSING=*value*] [, START=*array*]

QHULL - Constructs convex hulls, Delaunay triangulations, and Voronoi diagrams.
QHULL, *V*, *Tr* or QHULL, *V0* , *V1*, [, *V2* ... [, *V*6]] , *Tr* [, BOUNDS=*variable*] [, CONNECTIVITY=*variable*] [, /DELAUNAY] [, SPHERE=*variable*] [, VDIAGRAM=*array*] [, VNORMALS=*array*] [, VVERTICES=*array*]

QROMB - Evaluates integral over a closed interval.
Result = QROMB(*Func*, *A*, *B* [, /DOUBLE] [, EPS=*value*] [, JMAX=*value*] [, K=*value*])

QROMO - Evaluates integral over an open interval.
Result = QROMO(*Func*, *A* [, *B*] [, /DOUBLE] [, EPS=*value*] [, JMAX=*value*] [, K=*value*] [, /MIDEXP | , /MIDINF | , /MIDPNT | , /MIDSQL | , /MIDSQU])

QSIMP - Evaluates integral using Simpson's rule.
Result = QSIMP(*Func*, *A*, *B* [, /DOUBLE] [, EPS=*value*] [, JMAX=*value*])

QUERY_ASCII - Tests a file for compatibility with READ_ASCII.
Result = QUERY_ASCII (*Filename* [, *Info*])

QUERY_BMP - Obtains information about a BMP image file.
Result = QUERY_BMP (*Filename* [, *Info*])

QUERY_DICOM - Obtains information about a DICOM archive file.
Result = QUERY_DICOM(*Filename*[, *Info*] [, IMAGE_INDEX=*index*] [, /DICOMEX])

QUERY_GIF - Obtains information about a GIF image file.
Result = QUERY_GIF (*Filename* [, *Info*])

QUERY_IMAGE - Determines if a file is recognized as an image file.
Result = QUERY_IMAGE (*Filename*[, *Info*] [, CHANNELS=*variable*] [, DIMENSIONS=*variable*] [, HAS_PALETTE=*variable*] [, IMAGE_INDEX=*index*] [, NUM_IMAGES=*variable*] [, PIXEL_TYPE=*variable*] [, SUPPORTED_READ=*variable*] [, SUPPORTED_WRITE=*variable*] [, TYPE=*variable*])

QUERY_JPEG - Obtains information about a JPEG image file.
Result = QUERY_JPEG (*Filename* [, *Info*])

QUERY_JPEG2000 - Obtains information about a JPEG2000 image file.
Result = QUERY_JPEG2000(*Filename* [, *Info*])

QUERY_MRSID - Obtains information about a MrSID image file.
Result = QUERY_MRSID(*Filename* [, *Info*] [, LEVEL=*lvl*])

QUERY_PICT - Obtains information about a PICT image file.
Result = QUERY_PICT (*Filename*, *Info*)

QUERY_PNG - Obtains information about a PNG image file.
Result = QUERY_PNG (*Filename* [, *Info*])

QUERY_PPM - Obtains information about a PPM image file.
Result = QUERY_PPM (*Filename* [, *Info*] [, MAXVAL=*variable*])

QUERY_SRF - Obtains information about an SRF image file.
Result = QUERY_SRF (*Filename* [, *Info*])

QUERY_TIFF - Obtains information about a TIFF image file.
Result = QUERY_TIFF (*Filename* [, *Info*] [, GEOTIFF=*variable*] [, IMAGE_INDEX=*index*])

QUERY_WAV - Obtains information about a WAV sound file.
Result = QUERY_WAV (*Filename*[, *Info*])

R

R_CORRELATE - Computes rank correlation.
Result = R_CORRELATE(*X*, *Y* [, D=*variable*] [, /KENDALL] [, PROBD=*variable*] [, ZD=*variable*])

R_TEST - Runs test for randomness.
Result = R_TEST(*X* [, N0=*variable*] [, N1=*variable*] [, R=*variable*])

RADON - Returns the Radon transform of a two-dimensional image.
Radon Transform: *Result* = RADON(*Array*
[, /DOUBLE] [, DRHO=*scalar*] [, DX=*scalar*]
[, DY=*scalar*] [, /GRAY] [, /LINEAR] [, NRHO=*scalar*]
[, NTHETA=*scalar*] [, RHO=*variable*] [, RMIN=*scalar*]
[, THETA=*variable*] [, XMIN=*scalar*] [, YMIN=*scalar*])
Radon Backprojection: *Result* = RADON(*Array,*
/BACKPROJECT, RHO=*variable*, THETA=*variable*
[, /DOUBLE] [, DX=*scalar*] [, DY=*scalar*] [, /LINEAR]
[, NX=*scalar*] [, NY=*scalar*] [, XMIN=*scalar*]
[, YMIN=*scalar*])

RANDOMN - Returns normally-distributed pseudo-random numbers.
Result = RANDOMN(*Seed [, D_1 [, ...,D_8]]*
[[, BINOMIAL=*[trials, probability]*] [, /DOUBLE]
[, GAMMA=*integer*{>0}] [, /NORMAL]
[, POISSON=*value*] [, /UNIFORM] | [, /LONG]])

RANDOMU - Returns uniformly-distributed pseudo-random numbers.
Result = RANDOMU(*Seed [, D_1 [, ...,D_8]]*
[[, BINOMIAL=*[trials, probability]*] [, /DOUBLE]
[, GAMMA=*integer*{>0}] [, /NORMAL]
[, POISSON=*value*] [, /UNIFORM] | [, /LONG]])

RANKS - Computes magnitude-based ranks.
Result = RANKS(*X*)

RDPIX - Interactively displays image pixel values.
RDPIX, *Image* [, *X0*, *Y0*]

READ/READF - Reads formatted input from keyboard or file.
READ, [*Prompt,*] *Var_1, ..., Var_n*
READF, [*Prompt,*] *Unit, Var_1, ..., Var_n*
Keywords: [, AM_PM=*[string, string]*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, FORMAT=*value*] [, MONTHS=*string_array*{12 names}] [, PROMPT=*string*]

READ_ASCII - Reads data from an ASCII file.
Result = READ_ASCII([*Filename*]
[, COMMENT_SYMBOL=*string*] [, COUNT=*variable*]
[, DATA_START=*lines_to_skip*] [, DELIMITER=*string*]
[, HEADER=*variable*] [, MISSING_VALUE=*value*]
[, NUM_RECORDS=*value*] [, RECORD_START=*index*]
[, TEMPLATE=*value*] [, /VERBOSE])

READ_BINARY - Reads the contents of a binary file using a passed template or basic command line keywords.
Result = READ_BINARY ([*Filename* | *FileUnit*]
[, TEMPLATE=*template*] | [[, DATA_START=*value*]
[, DATA_TYPE=*typecodes*] [, DATA_DIMS=*array*]
[, ENDIAN=*string*]])

READ_BMP - Reads Microsoft Windows bitmap file (.BMP).
Result = READ_BMP(*Filename*, [, *R*, *G*, *B*] [, *Ihdr*]
[, /RGB])

READ_DICOM - Reads an image from a DICOM file.
Result = READ_DICOM (*Filename* [, *Red*, *Green*, *Blue*]
[, IMAGE_INDEX=*index*] [, /DICOMEX])

READ_GIF - Reads a GIF image.
READ_GIF, *Filename*, *Image* [, *R*, *G*, *B*] [, /CLOSE]
[, /MULTIPLE]

READ_IMAGE - Reads the image contents of a file and returns the image in an IDL variable.
Result = READ_IMAGE (*Filename* [, *Red*, *Green*, *Blue*]
[, IMAGE_INDEX=*index*])

READ_INTERFILE - Reads Interfile (v3.3) file.
READ_INTERFILE, *File*, *Data*

READ_JPEG - Reads JPEG file.
READ_JPEG [, *Filename* | , UNIT=*lun*] , *Image*
[, *Colortable*] [, BUFFER=*variable*]
[, COLORS=*value*{8 to 256}] [, DITHER={0 | 1 | 2}]
[, /GRAYSCALE] [, /ORDER] [, TRUE={1 | 2 | 3}]
[, /TWO_PASS_QUANTIZE]

READ_JPEG2000 - Reads a JPEG2000 file.
Result = READ_JPEG2000(*Filename* [, Red, Green, Blue]
[, DISCARD_LEVELS=*value*] [,
MAX_LAYERS=*value*] [, /ORDER]
[, REGION=[*StartX*, *StartY*, *Width*, *Height*]])

READ_MRSID - Reads MrSID file.
Result = READ_MRSID (*Filename* [, LEVEL=*lvl*]
[, SUB_RECT=*rect*])

READ_PICT - Reads Macintosh PICT (version 2) bitmap file.
READ_PICT, *Filename*, *Image* [, *R*, *G*, *B*]

READ_PNG - Reads Portable Network Graphics (PNG) file.
Result = READ_PNG (*Filename* [, *R*, *G*, *B*] [,/ORDER]
[, /VERBOSE] [, TRANSPARENT=*variable*])
or
READ_PNG, *Filename*, *Image* [, *R*, *G*, *B*] [,/ORDER]
[, /VERBOSE] [, TRANSPARENT=*variable*]

READ_PPM - Reads PGM (gray scale) or PPM (portable pixmap for color) file.
READ_PPM, *Filename*, *Image* [, MAXVAL=*variable*]

READ_SPR - Reads a row-indexed sparse matrix from a file.
Result = READ_SPR(*Filename*)

READ_SRF - Reads Sun Raster Format file.
READ_SRF, *Filename*, *Image* [, *R*, *G*, *B*]

READ_SYLK - Reads Symbolic Link format spreadsheet file.
Result = READ_SYLK(*File* [, /ARRAY]
[, /COLMAJOR] [, NCOLS=*columns*] [, NROWS=*rows*]
[, STARTCOL=*column*] [, STARTROW=*row*]
[, /USEDOUBLES] [, /USELONGS])

READ_TIFF - Reads TIFF format file.
Result = READ_TIFF(*Filename* [, *R*, *G*, *B*]
[, CHANNELS=*scalar or vector*]
[, DOT_RANGE=*variable*] [, GEOTIFF=*variable*]
[, IMAGE_INDEX=*value*] [, ICC_PROFILE=*variable*]
[, INTERLEAVE={0 | 1 |2}]
[, ORIENTATION=*variable*] [, PHOTOSHOP=*variable*]
[, PLANARCONFIG=*variable*] [, SUB_RECT=*[x, y, width, height]*] [, /UNSIGNED] [, /VERBOSE])

READ_WAV - Reads the audio stream from the named .WAV file.
Result = READ_WAV (*Filename* [, *Rate*])

READ_WAVE - Reads Wavefront Advanced Visualizer file.
READ_WAVE, *File*, *Variables*, *Names*, *Dimensions*
[, MESHNAMES=*variable*]

READ_X11_BITMAP - Reads X11 bitmap file.
READ_X11_BITMAP, *File*, *Bitmap* [, *X*, *Y*]
[, /EXPAND_TO_BYTES]

READ_XWD - Reads X Windows Dump file.
Result = READ_XWD(*Filename*[, *R*, *G*, *B*])

READS - Reads formatted input from a string variable.
READS, *Input*, *Var*$_1$, ..., *Var*$_n$ [, AM_PM=*[string, string]*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, FORMAT=*value*] [, MONTHS=*string_array*{12 names}]

READU - Reads unformatted binary data from a file.
READU, *Unit*, *Var*$_1$, ..., *Var*$_n$
[, TRANSFER_COUNT=*variable*]

REAL_PART - Returns the real part of a complex-valued argument.
Result = REAL_PART(*Z*)

REBIN - Resizes a vector or array by integer multiples.
Result = REBIN(*Array*, *D*$_1$ *[, ...,D*$_8$*]* [, /SAMPLE])

RECALL_COMMANDS - Returns entries in IDL's command recall buffer.
Result = RECALL_COMMANDS( )

RECON3 - Reconstructs a 3D representation of an object from 2D images.
Result = RECON3(*Images*, *Obj_Rot*, *Obj_Pos*, *Focal*, *Dist*,*Vol_Pos*, *Img_Ref*, *Img_Mag*, *Vol_Size* [, /CUBIC]
[, MISSING=*value*] [, MODE=*value*] [, /QUIET])

REDUCE_COLORS - Reduces the number of colors used in an image by eliminating unused pixel values.
REDUCE_COLORS, *Image*, *Values*

REFORM - Changes array dimensions without changing the total number of elements.
Result = REFORM(*Array*, *D*$_1$ *[, ...,D*$_8$*]*
[, /OVERWRITE])

REGION_GROW - Perform region growing.
Result = REGION_GROW(*Array*, *ROIPixels*
[, /ALL_NEIGHBORS] [, /NAN]
[, STDDEV_MULTIPLIER=*value* |
THRESHOLD=[*min*,*max*]])

REGISTER_CURSOR - Associates the given name with the given cursor information.
REGISTER_CURSOR, *Name*, *Image*[, MASK=*value*]
[, HOTSPOT=*value*] [, /OVERWRITE]

REGRESS - Computes fit using multiple linear regression.
Result = REGRESS(*X*, *Y*, [, CHISQ=*variable*]
[, CONST=*variable*] [, CORRELATION=*variable*]
[, /DOUBLE] [, FTEST=*variable*]
[, MCORRELATION=*variable*]
[, MEASURE_ERRORS=*vector*] [, SIGMA=*variable*]
[, STATUS=*variable*] [, YFIT=*variable*])

REPEAT...UNTIL - Repeats statement(s) until expression evaluates to true. Subject is always executed at least once.
REPEAT *statement* UNTIL *expression*
or
REPEAT BEGIN
statements
ENDREP UNTIL *expression*

REPLICATE - Creates an array of given dimensions, filled with specified value.
Result = REPLICATE(*Value*, *D*$_1$ *[, ...,D*$_8$*]* [, *Thread pool keywords*])

REPLICATE_INPLACE - Updates an array by replacing all or selected parts of it with a specified value.
REPLICATE_INPLACE, *X*, *Value* [, *D1*, *Loc1* [, *D2*, *Range*]] [, *Thread pool keywords*]

RESOLVE_ALL - Compiles any uncompiled routines.
RESOLVE_ALL [, CLASS=*string*]
[, /CONTINUE_ON_ERROR] [, /QUIET]

RESOLVE_ROUTINE - Compiles a routine.
RESOLVE_ROUTINE, *Name*
[, /COMPILE_FULL_FILE]
[, /EITHER | , /IS_FUNCTION] [, /NO_RECOMPLIE]

RESTORE - Restores IDL variables and routines saved in an IDL SAVE file.
RESTORE [, *Filename*] [, DESCRIPTION=*variable*]
[, FILENAME=*name*]
[, /RELAXED_STRUCTURE_ASSIGNMENT]
[, RESTORED_OBJECTS=*variable*] [, /VERBOSE]

RETALL - Returns control to the main program level.
RETALL

RETURN - Returns control to the next-higher program level.
RETURN [, *Return_value*]

REVERSE - Reverses the order of one dimension of an array.
Result = REVERSE(*Array* [, *Subscript_Index*]
[, /OVERWRITE])

RK4 - Solves differential equations using fourth-order Runge-Kutta method.
Result = RK4(*Y, Dydx, X, H, Derivs* [, /DOUBLE])

ROBERTS - Returns an approximation of Roberts edge enhancement.
Result = ROBERTS(*Image*)

ROT - Rotates an image by any amount.
Result = ROT(*A, Angle,* [*Mag, X_0, Y_0*] [, /INTERP]
[, CUBIC=*value*{-1 to 0}] [, MISSING=*value*]
[, /PIVOT])

ROTATE - Rotates/transposes an array in multiples of 90 degrees.
Result = ROTATE(*Array, Direction*)

ROUND - Rounds the argument to its closest integer.
Result = ROUND(*X* [, /L64] [, *Thread pool keywords*])

ROUTINE_INFO - Provides information about compiled procedures and functions.
Result = ROUTINE_INFO([*Routine*
[[, /PARAMETERS{must specify *Routine*}] [, /SOURCE
] [, /UNRESOLVED] [, /VARIABLES] | , /SYSTEM]]
[, /DISABLED] [, /ENABLED] [, /FUNCTIONS])

RS_TEST - Performs the Wilcoxon Rank-Sum test.
Result = RS_TEST(*X, Y* [, UX=*variable*] [, UY=*variable*])

S

S_TEST - Performs the Sign test.
Result = S_TEST(*X, Y* [, ZDIFF=*variable*])

SAVGOL - Returns coefficients of Savitzky-Golay smoothing filter.
Result = SAVGOL(*Nleft, Nright, Order, Degree*
[, /DOUBLE])

SAVE - Saves variables, system variables, and IDL routines in a file for later use.
SAVE [, *Var_1, ..., Var_n*] [, /ALL] [, /COMM,
/VARIABLES] [, /COMPRESS]
[, DESCRIPTION=*string*] [, /EMBEDDED]
[, FILENAME=*string*] [, /ROUTINES]
[, /SYSTEM_VARIABLES] [, /VERBOSE]

SCALE3 - Sets up axis ranges and viewing angles for 3D plots.
SCALE3 [, XRANGE=*vector*] [, YRANGE=*vector*]
[, ZRANGE=*vector*] [, AX=*degrees*] [, AZ=*degrees*]

SCALE3D - Scales 3D unit cube into the viewing area.
SCALE3D

SCOPE_LEVEL - Returns the scope level of the currently running procedure or function.
Result = SCOPE_LEVEL()

SCOPE_TRACEBACK - Returns the current interpreter call stack (the sequence of routine calls to the present point).
Result = SCOPE_TRACEBACK([/STRUCTURE]
[, /SYSTEM])

SCOPE_VARFETCH - Returns variables outside the local scope of the currently running procedure or function.
Variable = SCOPE_VARFETCH(*VarName*
[, COMMON=string] [, /ENTER] [, LEVEL=value]
[, /REF_EXTRA])

SCOPE_VARNAME - Returns the names of variables outside the local scope of the currently running procedure or function.
Result = SCOPE_VARNAME([*Var_1, ..., Var_n*] [, COMMON=string] [, COUNT=variable] [, LEVEL=value])

SEARCH2D - Finds “objects” or regions of similar data within a 2D array.
Result = SEARCH2D(*Array, Xpos, Ypos, Min_Val,*
Max_Val [, /DECREASE, /INCREASE
[, LPF_BAND=*integer*{≥3}]] [, /DIAGONAL])

SEARCH3D - Finds “objects” or regions of similar data values within a volume.
Result = SEARCH3D(*Array, Xpos, Ypos, Zpos, Min_Val,*
Max_Val [, /DECREASE, /INCREASE
[, LPF_BAND=*integer*{≥3}]] [, /DIAGONAL])

SET_PLOT - Sets the output device used by the IDL direct graphics procedures.
SET_PLOT, *Device* [, /COPY] [, /INTERPOLATE]

SET_SHADING - Sets the light source shading parameters.
SET_SHADING [, /GOURAUD] [, LIGHT=*[x, y, z]*]
[, /REJECT] [, VALUES=*[darkest, brightest]*]

SETENV - Adds or changes an environment variable.
SETENV, *Environment_Expression*

SFIT - Performs polynomial fit to a surface.
Result = SFIT(*Data, Degree*
[, /IRREGULAR, KX=*variable*, /MAX_DEGREE])

SHADE_SURF - Creates a shaded-surface representation of gridded data.
SHADE_SURF, *Z* [, *X, Y*] [, AX=*degrees*] [, AZ=*degrees*]
[, IMAGE=*variable*] [, MAX_VALUE=*value*]
[, MIN_VALUE=*value*] [, PIXELS=*pixels*] [, /SAVE]
[, SHADES=*array*] [, /XLOG] [, /YLOG]

SHADE_SURF - *continued*

Graphics Keywords: [, CHARSIZE=*value*] [, CHARTHICK=*integer*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, /NODATA] [, POSITION=*[X_0, Y_0, X_1, Y_1]*] [, SUBTITLE=*string*] [, /T3D] [, THICK=*value*] [, TICKLEN=*value*] [, TITLE=*string*] [, {X | Y | Z}CHARSIZE=*value*] [, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}] [, {X | Y | Z}MARGIN=*[left, right]*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=*[min, max]*] [, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*] [, {X | Y | Z}TICKFORMAT=*string*] [, {X | Y | Z}TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT=*scalar*] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string_array*] [, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKUNITS=*string*] [, {X | Y | Z}TICKV=*array*] [, {X | Y | Z}TICK_GET=*variable*] [, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

SHADE_SURF_IRR - Creates a shaded-surface representation of an irregularly gridded dataset.

SHADE_SURF_IRR, *Z*, *X*, *Y* [, AX=*degrees*] [, AZ=*degrees*] [, IMAGE=*variable*] [, PLIST=*variable*] [, /T3D]

SHADE_VOLUME - Contours a volume to create a list of vertices and polygons that can be displayed using POLYSHADE.

SHADE_VOLUME, *Volume*, *Value*, *Vertex*, *Poly* [, /LOW] [, SHADES=*array*] [, /VERBOSE] [, XRANGE=*vector*] [, YRANGE=*vector*] [, ZRANGE=*vector*]

SHIFT - Shifts elements of vectors or arrays by a specified number of elements.

Result = SHIFT(*Array*, S_1 *[, ..., S_n]*)

SHMDEBUG - Print debugging information when a variable loses reference to an underlying shared memory segment.

Result = SHMDEBUG(*Enable*)

SHMMAP - Maps anonymous shared memory, or local disk files, into the memory address space of the currently executing IDL process.

SHMMAP [, *SegmentName*] [, D_1, ..., D_8] [, /BYTE] [, /COMPLEX] [, /DCOMPLEX] [, /DESTROY_SEGMENT] [, DIMENSION=*value*] [, /DOUBLE] [, FILENAME=*value*] [, /FLOAT] [, GET_NAME=*value*] [, GET_OS_HANDLE=*value*] [, /INTEGER] [, /L64] [, /LONG] [, OFFSET=*value*] [, OS_HANDLE=*value*] [, /PRIVATE] [, SIZE=*value*] [, /SYSV] [, TEMPLATE=*value*] [, TYPE=*value*] [, /UINT] [, /UL64] [, /ULONG]

SHMUNMAP - Removes a memory segment previously created by SHMMAP from the system.

SHMUNMAP, *SegmentName*

SHMVAR - Creates an IDL array variable that uses the memory from a current mapped memory segment created by the SHMMAP procedure.

Result = SHMVAR(*SegmentName* [, D_1, ..., D_8] [, /BYTE] [, /COMPLEX] [, /DCOMPLEX] [, DIMENSION=*value*] [, /DOUBLE] [, /FLOAT] [, /INTEGER] [, /L64] [, /LONG] [, SIZE=*value*] [, TEMPLATE=*value*] [, TYPE=*value*] [, /UINT] [, /UL64] [, /ULONG])

SHOW3 - Displays array as image, surface plot, and contour plot simultaneously.

SHOW3, *Image* [, *X*, *Y*] [, /INTERP] [, E_CONTOUR=*structure*] [, E_SURFACE=*structure*] [, SSCALE=*scale*]

SHOWFONT - Displays a TrueType or vector font

SHOWFONT, *Font*, *Name* [, /ENCAPSULATED] [, /TT_FONT]

SIMPLEX - Use the simplex method to solve linear programming problems.

Result = SIMPLEX(*Zequation*, *Constraints*, M_1, M_2, M_3 [, *Tableau* [, *Izrov* [, *Iposv*]]] [, /DOUBLE] [, EPS = *value*] [, STATUS = *variable*])

SIN - Returns the trigonometric sine of *X*.

Result = SIN(*X* [, *Thread pool keywords*])

SINDGEN - Returns a string array with each element set to its subscript.

Result = SINDGEN(D_1 *[, ...,D_8]*)

SINH - Returns the hyperbolic sine of X.

Result = SINH(*X* [, *Thread pool keywords*])

SIZE - Returns array size and type information.

Result = SIZE(*Expression* [, /L64] [, /DIMENSIONS | , /FILE_LUN | , /FILE_OFFSET| , /N_DIMENSIONS | , /N_ELEMENTS | , /SNAME | , /STRUCTURE | , /TNAME | , /TYPE])

SKEWNESS - Computes statistical skewness of an *n*-element vector.

Result = SKEWNESS(*X* [, /DOUBLE] [, /NAN])

SKIP_LUN - Reads data in an open file and moves the file pointer.

SKIP_LUN, *FromUnit*, [, *Num*] [, /EOF] [, /LINES] [, /TRANSFER_COUNT=*variable*]

SLICER3 - Interactive volume visualization tool.

SLICER3 [, *hData3D*] [, DATA_NAMES=*string/string_array*] [, /DETACH] [, GROUP=*widget_id*] [, /MODAL]

SLIDE_IMAGE - Creates a scrolling graphics window for examining large images.
SLIDE_IMAGE [, *Image*] [, /BLOCK] [, CONGRID=0] [, FULL_WINDOW=*variable*] [, GROUP=*widget_id*] [, /ORDER] [, /REGISTER] [, RETAIN={0 | 1 | 2}] [, SLIDE_WINDOW=*variable*] [, SHOW_FULL=0] [, TITLE=*string*] [, TOP_ID=*variable*] [, XSIZE=*width*] [, XVISIBLE=*width*] [, YSIZE=*height*] [, YVISIBLE=*height*]

SMOOTH - Smooths with a boxcar average.
Result = SMOOTH(*Array, Width* [, /EDGE_TRUNCATE] [, MISSING=*value*] [, /NAN])

SOBEL - Returns an approximation of Sobel edge enhancement.
Result = SOBEL(*Image*)

SOCKET - Opens client-side TCP/IP Internet socket as IDL file unit.
SOCKET, *Unit*, *Host*, *Port* [, CONNECT_TIMEOUT=*value*] [, ERROR=*variable*] [, /GET_LUN] [, /RAWIO] [, READ_TIMEOUT=*value*] [, /SWAP_ENDIAN] [, /SWAP_IF_BIG_ENDIAN] [, /SWAP_IF_LITTLE_ENDIAN] [, WIDTH=*value*] [, WRITE_TIMEOUT=*value*]
UNIX-Only Keywords: [, /STDIO]

SORT - Returns indices of an array sorted in ascending order.
Result = SORT(*Array* [, /L64])

SPAWN - Spawns child process for access to operating system.
SPAWN [, *Command* [, *Result*] [, *ErrResult*]]
Keywords (all platforms): [, COUNT=*variable*] [, EXIT_STATUS=*variable*] [, PID=*variable*]
UNIX-Only Keywords: [, /NOSHELL] [, /NOTTYRESET] [, /NULL_STDIN] [, /SH] [, /STDERR] [, UNIT=*variable* {*Command* required, *Result* not allowed}]
Windows-Only Keywords: [, /HIDE] [, /LOG_OUTPUT] [, /NOSHELL] [, /NOWAIT] [, /NULL_STDIN] [, /STDERR]

SPH_4PNT - Returns center and radius of a sphere given 4 points.
SPH_4PNT, *X, Y, Z, Xc, Yc, Zc, R* [, /DOUBLE]

SPH_SCAT - Performs spherical gridding.
Result = SPH_SCAT(*Lon, Lat, F* [, BOUNDS=*[lonmin, latmin, lonmax, latmax]*] [, BOUT=*variable*] [, GOUT=*variable*] [, GS=*[lonspacing, latspacing]*] [, NLON=*value*] [, NLAT=*value*])

SPHER_HARM - Returns value of the spherical harmonic function.
Result=SPHER_HARM(*Theta, Phi, L, M,* [, /DOUBLE])

SPL_INIT - Establishes the type of interpolating spline.
Result = SPL_INIT(*X, Y* [, /DOUBLE] [, YP0=*value*] [, YPN_1=*value*])

SPL_INTERP - Performs cubic spline interpolation.
Result = SPL_INTERP(*X, Y, Y2, X2* [, /DOUBLE])

SPLINE - Performs cubic spline interpolation.
Result = SPLINE(*X, Y, T* [, *Sigma*] [, /DOUBLE])

SPLINE_P - Performs parametric cubic spline interpolation.
SPLINE_P, *X, Y, Xr, Yr* [, /DOUBLE] [, INTERVAL=*value*] [, TAN0=*[X_0, Y_0]*] [, TAN1=*[X_{n-1}, Y_{n-1}]*]

SPRSAB - Performs matrix multiplication on sparse matrices.
Result = SPRSAB(*A, B* [, /DOUBLE] [, THRESHOLD=*value*])

SPRSAX - Multiplies sparse matrix by a vector.
Result = SPRSAX(*A, X* [, /DOUBLE])

SPRSIN - Converts matrix to row-index sparse matrix.
Result = SPRSIN(*A* [, /COLUMN] [, /DOUBLE] [, THRESHOLD=*value*]) or
Result = SPRSIN(*Columns, Rows, Values, N* [, /DOUBLE] [, THRESHOLD=*value*])

SPRSTP - Constructs the transpose of a sparse matrix.
Result = SPRSTP(*A*)

SQRT - Returns the square root of *X*.
Result = SQRT(*X* [, *Thread pool keywords*])

STANDARDIZE - Computes standardized variables.
Result = STANDARDIZE(*A* [, /DOUBLE])

STDDEV - Computes the standard deviation of an *n*-element vector.
Result = STDDEV(*X* [, /DOUBLE] [, /NAN])

STOP - Stops the execution of a running program or batch file.
STOP [, *$Expr_1$, ..., $Expr_n$*]

STRARR - Returns string array containing zero-length strings.
Result = STRARR(*D_1 [, ...,D_8]*)

STRCMP - Compares two strings.
Result = STRCMP(*String1, String2* [, *N*] [, /FOLD_CASE])

STRCOMPRESS - Removes whitespace from a string.
Result = STRCOMPRESS(*String* [, /REMOVE_ALL])

STREAMLINE - Generates the visualization graphics from a path.
STREAMLINE, *Verts, Conn, Normals, Outverts, Outconn* [, ANISOTROPY=*array*] [, SIZE=*vector*] [, PROFILE=*array*]

STREGEX - Performs regular expression matching.
Result = STREGEX(*StringExpression, RegularExpression* [, /BOOLEAN | , /EXTRACT | , LENGTH=*variable* [, /SUBEXPR]] [, /FOLD_CASE])

STRETCH - Stretches color table for contrast enhancement.
STRETCH [, *Low, High* [, *Gamma*]] [, /CHOP]

STRING - Converts its arguments to string type.
Result = STRING(*Expression*$_1$, ..., *Expression*$_n$ [, AM_PM=*[string, string]*] [, DAYS_OF_WEEK=*string_array*{7 names}] [, FORMAT=*value*] [, MONTHS=*string_array*{12 names}] [, /PRINT])

STRJOIN - Collapses a string scalar or array into merged strings.
Result = STRJOIN(*String* [, *Delimiter*] [, /SINGLE])

STRLEN - Returns the length of a string.
Result = STRLEN(*Expression*)

STRLOWCASE - Converts a string to lower case.
Result = STRLOWCASE(*String*)

STRMATCH - Compares search string against input string expression.
Result = STRMATCH(*String*, *SearchString* [, /FOLD_CASE])

STRMESSAGE - Returns the text of an error number.
Result = STRMESSAGE(*Err* [, /BLOCK | , /CODE | , /NAME])

STRMID - Extracts a substring from a string.
Result = STRMID(*Expression*, *First_Character* [, *Length*] [, /REVERSE_OFFSET])

STRPOS - Finds first occurrence of a substring within a string.
Result = STRPOS(*Expression*, *Search String* [, *Pos*] [, /REVERSE_OFFSET] [, /REVERSE_SEARCH])

STRPUT - Inserts the contents of one string into another.
STRPUT, *Destination*, *Source* [, *Position*]

STRSPLIT - Splits its input string argument into separate substrings, according to the specified pattern.
Result = STRSPLIT(*String* [, *Pattern*] [, COUNT=*variable*] [, ESCAPE=*string* | , /REGEX [, /FOLD_CASE]] [, /EXTRACT | , LENGTH=*variable*] [, /PRESERVE_NULL])

STRTRIM - Removes leading and/or trailing blanks from string.
Result = STRTRIM(*String* [, *Flag*])

STRUCT_ASSIGN - Performs “relaxed structure assignment” to copy a structure.
STRUCT_ASSIGN, *Source*, *Destination* [, /NOZERO] [, /VERBOSE]

STRUCT_HIDE - Prevents the IDL HELP procedure from displaying information about structures or objects.
STRUCT_HIDE, *Arg*$_1$ [, *Arg*$_2$, ..., *Arg*$_n$]

STRUPCASE - Converts a string to upper case.
Result = STRUPCASE(*String*)

SURFACE - Plots an array as a wireframe mesh surface.
SURFACE, *Z* [, *X*, *Y*] [, AX=*degrees*] [, AZ=*degrees*] [, BOTTOM=*index*] [, /HORIZONTAL] [, /LEGO] [, /LOWER_ONLY | , /UPPER_ONLY] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, /SAVE] [, SHADES=*array*] [, SKIRT=*value*] [, /XLOG] [, /YLOG] [, ZAXIS={1 | 2 | 3 | 4}] [, /ZLOG]
Graphics Keywords: Accepts all graphics keywords accepted by PLOT except for: PSYM, SYMSIZE.

SURFR - Sets up 3D transformations by duplicating rotation, translation, and scaling of SURFACE.
SURFR [, AX=*degrees*] [, AZ=*degrees*]

SVDC - Computes Singular Value Decomposition of an array.
SVDC, *A*, *W*, *U*, *V* [, /COLUMN] [, /DOUBLE] [, ITMAX=*value*]

SVDFIT - Multivariate least squares fit using SVD method.
Result = SVDFIT(*X*, *Y* [, *M*] [, A=*vector*] [, CHISQ=*variable*] [, COVAR=*variable*] [, /DOUBLE] [, FUNCTION_NAME=*string*] [, /LEGENDRE] [, MEASURE_ERRORS=*vector*] [, SIGMA=*variable*] [, SING_VALUES=*variable*] [, SINGULAR=*variable*] [, STATUS=*variable*] [, TOL=*value*] [, VARIANCE=*variable*] [, YFIT=*variable*])

SVSOL - Solves set of linear equations using back-substitution.
Result = SVSOL(*U*, *W*, *V*, *B* [, /COLUMN] [, /DOUBLE])

SWAP_ENDIAN - Reverses the byte ordering of scalars, arrays or structures.
Result = SWAP_ENDIAN(*Variable* [, /SWAP_IF_BIG_ENDIAN] [, /SWAP_IF_LITTLE_ENDIAN])

SWAP_ENDIAN_INPLACE - Reverses the byte ordering of scalars, arrays or structures.
SWAP_ENDIAN_INPLACE, *Variable* [, /SWAP_IF_BIG_ENDIAN] [, /SWAP_IF_LITTLE_ENDIAN]

SWITCH - Selects one statement for execution from multiple choices, depending upon the value of an expression.
SWITCH expression OF
    expression: statement
expression: statement
ELSE: statement
ENDSWITCH

SYSTIME - Returns the current time as either a string, as the number of seconds elapsed since 1 January 1970, or as a Julian d value.
String = SYSTIME([0 [, *ElaspedSeconds*]] [, /UTC])
or
Seconds = SYSTIME(1 | /SECONDS)
or
Julian = SYSTIME(/JULIAN [, /UTC])

T

T_CVF - Computes the cutoff value in a Student's t distribution.
Result = T_CVF(*P, Df*)

T_PDF - Computes Student's t distribution.
Result = T_PDF(*V, Df*)

T3D - Performs various 3D transformations.
T3D [, *Array* | , /RESET] [, MATRIX=*variable*] [, OBLIQUE=*vector*] [, PERSPECTIVE=*p*{eye at (0,0,*p*)}] [, ROTATE=*[x, y, z]*] [, SCALE=*[x, y, z]*] [, TRANSLATE=*[x, y, z]*] [, /XYEXCH | , /XZEXCH | , /YZEXCH]

TAG_NAMES - Returns the names of tags in a structure.
Result = TAG_NAMES(*Expression* [, /STRUCTURE_NAME])

TAN - Returns the tangent of *X*.
Result = TAN(*X* [, *Thread pool keywords*])

TANH - Returns the hyperbolic tangent of X.
Result = TANH(*X* [, *Thread pool keywords*])

TEK_COLOR - Loads color table based on Tektronix printer.
TEK_COLOR [, *Start_Index*, *Colors*]

TEMPORARY - Returns a temporary copy of a variable, and sets the original variable to "undefined".
Result = TEMPORARY(*Variable*)

TETRA_CLIP - Clips a tetrahedral mesh to an arbitrary plane in space and returns a tetrahedral mesh of the remaining portion.
Result = TETRA_CLIP (*Plane*, *Vertsin*, *Connin*, *Vertsout*, *Connout* [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*] [, CUT_VERTS=*variable*])

TETRA_SURFACE - Extracts a polygonal mesh as the exterior surface of a tetrahedral mesh.
Result = TETRA_SURFACE (*Verts*, *Connin*)

TETRA_VOLUME - Computes properties of tetrahedral mesh array.
Result = TETRA_VOLUME (*Verts*, *Conn* [, AUXDATA=*array*] [, MOMENT=*variable*])

THIN - Returns the "skeleton" of a bi-level image.
Result = THIN(*Image*[, /NEIGHBOR_COUNT] [, /PRUNE])

THREED - Plots a 2D array as a pseudo 3D plot.
THREED, *A* [, *Sp*] [, TITLE=*string*] [, XTITLE=*string*] [, YTITLE=*string*]

TIME_TEST2 - Performs speed benchmarks for IDL.
TIME_TEST2 [, *Filename*]

TIMEGEN - Returns an array of double-precision floating-point values that represent times in Julian values.
Result = TIMEGEN([D1,...,D8 | , FINAL=*value*] [, DAYS=*vector*] [, HOURS=*vector*] [, MINUTES=*vector*] [, MONTHS=*vector*] [, SECONDS=*vector*] [, START=*value*] [, STEP_SIZE=*value*] [, UNITS=*string*] [, YEAR=*value*])

TM_TEST - Performs t-means test.
Result = TM_TEST(*X*, *Y* [, /PAIRED] [, /UNEQUAL])

TOTAL - Sums of the elements of an array.
Result = TOTAL(*Array* [, *Dimension*] [, /CUMULATIVE] [, /DOUBLE] [, /INTEGER] [, /NAN] [, /PRESERVE_TYPE] [, *Thread pool keywords*])

TRACE - Computes the trace of an array.
Result = TRACE(*A* [, /DOUBLE])

TrackBall Object - See "TrackBall" on page 112.

TRANSPOSE - Transposes an array.
Result = TRANSPOSE(*Array* [, *P*])

TRI_SURF - Interpolates gridded set of points with a smooth quintic surface.
Result = TRI_SURF(*Z* [, *X*, *Y*] [, /EXTRAPOLATE] [, MISSING=*value*] [, /REGULAR] [, XGRID=*[xstart, xspacing]* | [, XVALUES=*array*]] [, YGRID=*[yxstart, yspacing]* | [, YVALUES=*array*]] [, GS=*[xspacing, yspacing]*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*] [, NX=*value*] [, NY=*value*])

TRIANGULATE - Constructs Delaunay triangulation of a planar set of points.
TRIANGULATE, *X, Y, Triangles* [, *B*] [, CONNECTIVITY=*variable*] [, SPHERE=*variable* [, /DEGREES]] [, FVALUE=*variable*] [, REPEATS=*variable*] [, TOLERANCE=*value*]

TRIGRID - Interpolates irregularly-gridded data to a regular grid.
Result = TRIGRID(*X, Y, Z, Triangles* [, *GS, Limits*])
For spherical gridding:
Result = TRIGRID(*F , GS, Limits*, SPHERE=*S*)
Keywords: [, /DEGREES] [, EXTRAPOLATE=*array* | , /QUINTIC] [, INPUT=*variable*] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, MISSING=*value*] [, NX=*value*] [, NY=*value*] [, SPHERE=*variable*] [, XGRID=*variable*] [, YGRID=*variable*] [, XOUT=*vector*, YOUT=*vector*]

TRIQL - Determines eigenvalues and eigenvectors of tridiagonal array.
TRIQL, *D, E, A* [, /DOUBLE]

TRIRED - Reduces a real, symmetric array to tridiagonal form.
TRIRED, *A, D, E* [, /DOUBLE]

TRISOL - Solves tridiagonal systems of linear equations.
Result = TRISOL(*A*, *B*, *C*, *R* [, /DOUBLE])

TRUNCATE_LUN - Truncates an open file at the location of the current file pointer.
TRUNCATE_LUN, $Unit_1$, ..., $Unit_n$

TS_COEF - Computes the coefficients for autoregressive time-series.
Result = TS_COEF(*X*, *P* [, /DOUBLE] [, MSE=*variable*])

TS_DIFF - Computes the forward differences of a time-series.
Result = TS_DIFF(*X*, *K* [, /DOUBLE])

TS_FCAST - Computes future or past values of a stationary time-series.
Result = TS_FCAST(*X*, *P*, *Nvalues* [, /BACKCAST] [, /DOUBLE])

TS_SMOOTH - Computes moving averages of a time-series.
Result = TS_SMOOTH(*X*, *Nvalues* [, /BACKWARD] [, /DOUBLE] [, /FORWARD] [, ORDER=*value*])

TV - Displays an image.
TV, *Image* [, *Position*]
or
TV, *Image* [, *X*, *Y* [, *Channel*]]
Keywords: [, /CENTIMETERS | , /INCHES] [, /ORDER] [, TRUE={1 | 2 | 3}] [, /WORDS] [, XSIZE=*value*] [, YSIZE=*value*]
Graphics Keywords: [, CHANNEL=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, /T3D | Z=*value*]

TVCRS - Manipulates the image display cursor.
TVCRS [, *ON_OFF*]
or
TVCRS [, *X*, *Y*]
Keywords: [, /CENTIMETERS | , /INCHES] [, /HIDE_CURSOR]
Graphics Keywords: [, /DATA | , /DEVICE | , /NORMAL] [, /T3D | Z=*value*]

TVLCT - Loads display color tables.
TVLCT, V_1, V_2, V_3 [, *Start*] [, /GET] [, /HLS | , /HSV]
or
TVLCT, *V* [, *Start*] [, /GET] [, /HLS | , /HSV]

TVRD - Reads an image from a window into a variable.
Result = TVRD([X_0 [, Y_0 [, N_x [, N_y [, *Channel*]]]]] [, CHANNEL=*value*] [, /ORDER] [, TRUE={1 | 2 | 3}] [, /WORDS])

TVSCL - Scales and displays an image.
TVSCL, *Image* [, *Position*]
or
TVSCL, *Image* [, *X*, *Y* [, *Channel*]]
Keywords: [, /CENTIMETERS | , /INCHES] [, /NAN] [, /ORDER] [, TOP=*value*] [, TRUE={1 | 2 | 3}] [, /WORDS] [, XSIZE=*value*] [, YSIZE=*value*]
Graphics Keywords: [, CHANNEL=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, /T3D | Z=*value*] [, *Thread pool keywords*]

U

UINDGEN - Returns unsigned integer array with each element set to its subscript.
Result = UINDGEN(D_1 [, ...,D_8] [, *Thread pool keywords*])

UINT - Converts argument to unsigned integer type.
Result = UINT(*Expression* [, *Offset* [, D_1 [, ...,D_8]]] [, *Thread pool keywords*])

UINTARR - Returns an unsigned integer vector or array.
Result = UINTARR(D_1 [, ...,D_8] [, /NOZERO])

UL64INDGEN - Returns an unsigned 64-bit integer array with each element set to its subscript.
Result = UL64INDGEN(D_1 [, ...,D_8] [, *Thread pool keywords*])

ULINDGEN - Returns an unsigned longword array with each element set to its subscript.
Result = ULINDGEN(D_1 [, ...,D_8] [, *Thread pool keywords*])

ULON64ARR - Returns an unsigned 64-bit integer vector or array.
Result = ULON64ARR(D_1 [, ...,D_8] [, /NOZERO])

ULONARR - Returns an unsigned longword integer vector or array.
Result = ULONARR(D_1 [, ...,D_8] [, /NOZERO])

ULONG - Converts argument to unsigned longword integer type.
Result = ULONG(*Expression* [, *Offset* [, D_1 [, ...,D_8]]] [, *Thread pool keywords*])

ULONG64 - Converts argument to unsigned 64-bit integer type.
Result = ULONG64(*Expression* [, *Offset* [, D_1 [, ...,D_8]]] [, *Thread pool keywords*])

UNIQ - Returns subscripts of the unique elements in an array.
Result = UNIQ(*Array* [, *Index*])

UNSHARP_MASK - Performs an unsharp-mask sharpening filter on a two-dimensional array or a TrueColor image.
Result = UNSHARP_MASK(*Image* [, AMOUNT=*value*] [, RADIUS=*value*] [, THRESHOLD=*value*] [, TRUE={1 | 2 | 3}])

USERSYM - Defines a new plotting symbol.
USERSYM, *X* [, *Y*] [, COLOR=*value*] [, /FILL] [, THICK=*value*]

V

VALUE_LOCATE - Finds the intervals within a given monotonic vector that brackets a given set of one or more search values.
Result = VALUE_LOCATE (*Vector*, *Value* [, /L64])

VARIANCE - Computes the statistical variance of an *n*-element vector.
Result = VARIANCE(*X* [, /DOUBLE] [, /NAN])

VECTOR_FIELD - Places colored, oriented vectors of specified length at each vertex in an input vertex array.
VECTOR_FIELD, *Field*, *Outverts*, *Outconn*
[, ANISOTROPY=*array*] [, SCALE=*value*]
[, VERTICES=*array*]

VEL - Draws a velocity (flow) field with streamlines.
VEL, *U*, *V* [, NVECS=*value*] [, XMAX=
value{*xsize*/*ysize*}] [, LENGTH=*value*{*longest*/*steps*}]
[, NSTEPS=*value*] [, TITLE=*string*]

VELOVECT - Draws a 2D velocity field plot.
VELOVECT, *U*, *V* [, *X*, *Y*] [, COLOR=*index*]
[, MISSING=*value* [, /DOTS]] [, LENGTH=*value*]
[, /OVERPLOT] [Also accepts all PLOT keywords]

VERT_T3D - Transforms a 3D array by a 4x4 transformation matrix.
Result = VERT_T3D(*Vertex_List* [, DOUBLE=*value*]
[, MATRIX=*4x4_array*] [, /NO_COPY] [, /NO_DIVIDE
[, SAVE_DIVIDE=*variable*]])

VOIGT - Calculates intensity of atomic absorption line (Voigt) profile.
Result = VOIGT(*A*, *U* [, *Thread pool keywords*])

VORONOI - Computes Voronoi polygon given Delaunay triangulation.
VORONOI, *X*, *Y*, *I0*, *C*, *Xp*, *Yp*, *Rect*

VOXEL_PROJ - Creates volume visualizations using voxel technique.
Result = VOXEL_PROJ(*V* [, *RGBO*]
[, BACKGROUND=*array*]
[, CUTTING_PLANE=*array*] [, /INTERPOLATE]
[, /MAXIMUM_INTENSITY] [, STEP=*[Sx, Sy, Sz]*]
[, XSIZE=*pixels*] [, YSIZE=*pixels*]
[, ZBUFFER=*int_array*] [, ZPIXELS=*byte_array*])

W

WAIT - Suspends execution of an IDL program for a specified period.
WAIT, *Seconds*

WARP_TRI - Warps an image using control points.
Result = WARP_TRI(*Xo*, *Yo*, *Xi*, *Yi*, *Image*
[, /EXTRAPOLATE] [, OUTPUT_SIZE=*vector*]
[, /QUINTIC] [, /TPS])

WATERSHED - Applies the morphological watershed operator to a grayscale image.
Result = WATERSHED (*Image*
[, CONNECTIVITY={4 | 8}] [, /LONG]
[, NREGIONS=*variable*])

WDELETE - Deletes IDL graphics windows.
WDELETE [, *Window_Index* [, ...]]

WF_DRAW - Draws weather fronts with smoothing.
WF_DRAW, *X*, *Y* [[, /COLD | , FRONT_TYPE=1] |
[, /WARM | , FRONT_TYPE=2] | [, /OCCLUDED | ,
FRONT_TYPE=3] | [, /STATIONARY | ,
FRONT_TYPE=4] | [, /CONVERGENCE |
FRONT_TYPE=5]] [, COLOR=*value*] [, /DATA | ,
/DEVICE | , /NORMAL] [, INTERVAL=*value*]
[, PSYM=*value*] [, SYM_HT=*value*]
[, SYM_LEN=*value*] [, THICK=*value*]

WHERE - Returns subscripts of nonzero array elements.
Result = WHERE(*Array_Expression* [, *Count*]
[, COMPLEMENT=*variable*] [, /L64]
[, NCOMPLEMENT=*variable*] [, *Thread pool keywords*])

WHILE...DO - Performs statement(s) as long as expression evaluates to true. Subject is never executed if condition is initially false.
WHILE *expression* DO *statement*
or
WHILE *expression* DO BEGIN
statements
ENDWHILE

WIDGET_ACTIVEX - Create an ActiveX control and place it into an IDL widget hierarchy.
Result = WIDGET_ACTIVEX(*Parent*, *COM_ID*,
[, /ALIGN_BOTTOM | , /ALIGN_CENTER | ,
/ALIGN_LEFT | , /ALIGN_RIGHT | , /ALIGN_TOP]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FUNC_GET_VALUE=*string*] [ID_TYPE=*value*]
[, KILL_NOTIFY=*string*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, XOFFSET=*value*]
[, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_BASE - Creates base widget (containers for other widgets).
Result = WIDGET_BASE([*Parent*] [, /ALIGN_BOTTOM
| , /ALIGN_CENTER | , /ALIGN_LEFT | ,
/ALIGN_RIGHT | , /ALIGN_TOP] [, MBAR=variable |
, /MODAL] [, /BASE_ALIGN_BOTTOM | ,
/BASE_ALIGN_CENTER | , /BASE_ALIGN_LEFT | ,
/BASE_ALIGN_RIGHT | , /BASE_ALIGN_TOP]
[, COLUMN=*ncols* | , ROW=*nrows*]
[, /CONTEXT_EVENTS] [, /CONTEXT_MENU]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, /EXCLUSIVE | , /NONEXCLUSIVE] [, /FLOATING]
[, FRAME=*width*] [, FUNC_GET_VALUE=*string*]
[, /GRID_LAYOUT]
[, GROUP_LEADER=*widget_id*{must specify for modal dialogs}] [, /KBRD_FOCUS_EVENTS]
[, KILL_NOTIFY=*string*] [, /MAP{not for modal bases}]
[, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SCROLL{not for modal bases}] [, /SENSITIVE] [, SPACE=*value*{ignored if exclusive or nonexclusive}] [, TAB_MODE=*value*]

WIDGET_BASE - *continued*
[, TITLE=*string*] [, TLB_FRAME_ATTR=*value*{top-level bases only}] [, /TLB_ICONIFY_EVENTS{top-level bases only}] [, /TLB_KILL_REQUEST_EVENTS{top-level bases only}] [, /TLB_MOVE_EVENTS{top-level bases only}] [, /TLB_SIZE_EVENTS{top-level bases only}] [, /TOOLBAR] [, /TRACKING_EVENTS] [, UNAME=*string*] [, UNITS={0 | 1 | 2}] [, UVALUE=*value*] [, XOFFSET=*value*] [, XPAD=*value*{ignored if exclusive or nonexclusive}] [, XSIZE=*value*] [, X_SCROLL_SIZE=*value*] [, YOFFSET=*value*] [, YPAD=*value*{ignored if exclusive or nonexclusive}] [, YSIZE=*value*] [, Y_SCROLL_SIZE=*value*])
X Windows Keywords: [, DISPLAY_NAME=*string*] [, RESOURCE_NAME=*string*] [, RNAME_MBAR=*string*]

WIDGET_BUTTON - Creates button widgets.
Result = WIDGET_BUTTON(*Parent* [, /ACCELERATOR=*string*] [, /ALIGN_CENTER | , /ALIGN_LEFT | , /ALIGN_RIGHT] [, /BITMAP] [, /CHECKED_MENU] [, /DYNAMIC_RESIZE] [, EVENT_FUNC=*string*] [, EVENT_PRO=*string*] [, FONT=*string*] [, FRAME=*width*] [, FUNC_GET_VALUE=*string*] [, GROUP_LEADER=*widget_id*] [, /HELP] [, KILL_NOTIFY=*string*] [, /MENU] [, /NO_COPY] [, /NO_RELEASE] [, NOTIFY_REALIZE=*string*] [, PRO_SET_VALUE=*string*] [, /PUSHBUTTON_EVENTS] [, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*] [, /SENSITIVE] [, /SEPARATOR] [, TAB_MODE=*value*] [, TOOLTIP=*string*] [, /TRACKING_EVENTS] [, UNAME=*string*] [, UNITS={0 | 1 | 2}] [, UVALUE=*value*] [, VALUE=*value*] [, X_BITMAP_EXTRA=*bits*] [, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])
X Windows Keywords: [, RESOURCE_NAME=*string*]

WIDGET_COMBOBOX - Creates editable droplist widgets.
Result = WIDGET_COMBOBOX(*Parent* [, /DYNAMIC_RESIZE] [, /EDITABLE] [, EVENT_FUNC=*string*] [, EVENT_PRO=*string*] [, FONT=*string*] [, FRAME=*value*] [, FUNC_GET_VALUE=*string*] [, GROUP_LEADER=*widget_id*] [, IGNORE_ACCELERATORS=*value*] [, KILL_NOTIFY=*string*] [, /NO_COPY] [, NOTIFY_REALIZE=*string*] [, PRO_SET_VALUE=*string*] [, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*] [, /SENSITIVE] [, TAB_MODE=*value*] [, /TRACKING_EVENTS] [, UNAME=*string*] [, UNITS={0 | 1 | 2}] [, UVALUE=*value*] [, VALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_CONTROL - Realizes, manages, and destroys widgets.
WIDGET_CONTROL [, *Widget_ID*]
All widgets: [, BAD_ID=*variable*] [, /CLEAR_EVENTS] [, DEFAULT_FONT=*string*{do not specify *Widget_ID*}] [, /DELAY_DESTROY{do not specify *Widget_ID*}] [, /DESTROY] [, EVENT_FUNC=*string*] [, EVENT_PRO=*string*] [, FUNC_GET_VALUE=*string*] [, GET_UVALUE=*variable*] [, GROUP_LEADER=*widget_id*] [, /HOURGLASS{do not specify *Widget_ID*}] [, KILL_NOTIFY=*string*] [, /MAP] [, /NO_COPY] [, NOTIFY_REALIZE=*string*] [, PRO_SET_VALUE=*string*] [, /PUSHBUTTON_EVENTS] [, /REALIZE] [, /RESET{do not specify *Widget_ID*}] [, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*] [, SEND_EVENT=*structure*] [, /SENSITIVE] [, SET_UNAME=*string*] [, SET_UVALUE=*value*] [, /SHOW] [, TIMER=*value*] [, TLB_GET_OFFSET=*variable*] [, TLB_GET_SIZE=*variable*] [, /TLB_KILL_REQUEST_EVENTS] [, TLB_SET_TITLE=*string*] [, TLB_SET_XOFFSET=*value*] [, TLB_SET_YOFFSET=*value*] [, /TRACKING_EVENTS] [, UNITS={0 | 1 | 2}] [, /UPDATE] [, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*]
WIDGET_BASE: [, /CONTEXT_EVENTS] [, /ICONIFY] [, /KBRD_FOCUS_EVENTS] [, TAB_MODE=*value*] [, /TLB_ICONIFY_EVENTS] [, /TLB_KILL_REQUEST_EVENTS] [, /TLB_MOVE_EVENTS] [, /TLB_SIZE_EVENTS]
WIDGET_BUTTON: [, /BITMAP] [, /DYNAMIC_RESIZE] [, GET_VALUE=*value*] [, /INPUT_FOCUS] [, /PUSHBUTTON_EVENTS] [, /SET_BUTTON] [, SET_VALUE=*value*] [, TAB_MODE=*value*] [, TOOLTIP=*string*] [, X_BITMAP_EXTRA=*bits*]
WIDGET_COMBOBOX: [, COMBOBOX_ADDITEM=*string*] [, COMBOBOX_DELETEITEM=*integer*] [, COMBOBOX_INDEX=*integer*] [, /DYNAMIC_RESIZE] [, GET_VALUE=*value*] [, IGNORE_ACCELERATORS={*string_array*|{0 | 1}}] [, SET_COMBOBOX_SELECT=*integer*] [, SET_VALUE=*value*] [, TAB_MODE=*value*]

WIDGET_CONTROL - *continued*

WIDGET_DRAW: [, /DRAW_BUTTON_EVENTS]
[, /DRAW_EXPOSE_EVENTS]
[, DRAW_KEYBOARD_EVENTS={0 | 1 | 2}]
[, /DRAW_MOTION_EVENTS]
[, /DRAW_VIEWPORT_EVENTS]
[, /DRAW_WHEEL_EVENTS]
[, DRAW_XSIZE=*integer*] [, DRAW_YSIZE=*integer*]
[, GET_DRAW_VIEW=*variable*]
[, GET_UVALUE=*variable*] [, GET_VALUE=*variable*]
[, IGNORE_ACCELERATORS={*string_array*|{0 | 1}}]
[, /INPUT_FOCUS] [, SET_DRAW_VIEW=*[x, y]*]
[, TOOLTIP=*string*]

WIDGET_DROPLIST: [, /DYNAMIC_RESIZE]
[, GET_VALUE=*value*]
[, SET_DROPLIST_SELECT=*integer*]
[, SET_VALUE=*value*] [, TAB_MODE=*value*]

WIDGET_LABEL: [, /DYNAMIC_RESIZE]
[, GET_VALUE=*value*] [, SET_VALUE=*value*]

WIDGET_LIST: [, /CONTEXT_EVENTS]
[, SET_LIST_SELECT=*value*]
[, SET_LIST_TOP=*integer*] [, SET_VALUE=*value*]
[, TAB_MODE=*value*]

WIDGET_PROPERTYSHEET:
[, /CONTEXT_EVENTS] [, /EDITABLE]
[, IGNORE_ACCELERATORS={*string_array*|{0 | 1}}]
[, /MULTIPLE_PROPERTIES]
[, PROPERTYSHEET_SETSELECTED=*empty string*, *string*, or *array of strings*]
[, REFRESH_PROPERTY=*string*, *string array*, or *integer*] [, SET_VALUE=*value*]

WIDGET_SLIDER: [, GET_VALUE=*value*]
[, SET_SLIDER_MAX=*value*]
[, SET_SLIDER_MIN=*value*] [, SET_VALUE=*value*]
[, TAB_MODE=*value*]

WIDGET_TAB: [, SET_TAB_CURRENT=*index*]
[, SET_TAB_MULTILINE=*value*]
[, TAB_MODE=*value*]

WIDGET_TABLE: [, ALIGNMENT={0 | 1 | 2}]
[, /ALL_TABLE_EVENTS] [, AM_PM=*[string, string]*]
[, BACKGROUND_COLOR=*array*]
[, COLUMN_LABELS=*string_array*]
[, COLUMN_WIDTHS=*array*]
[, /CONTEXT_EVENTS]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, /DELETE_COLUMNS{not for row_major mode}]
[, /DELETE_ROWS{not for column_major mode}]
[, /DISJOINT_SELECTION] [, /EDITABLE]
[, EDIT_CELL=*[integer, integer]*]
[, FONT=*string array*]
[, FOREGROUND_COLOR=*array*] [, FORMAT=*value*]
[, GET_VALUE=*variable*]
[, IGNORE_ACCELERATORS={*string_array*|{0 | 1}}]
[, INSERT_COLUMNS=*value*]
[, INSERT_ROWS=*value*]
[, /KBRD_FOCUS_EVENTS]
[, MONTHS=*string_array*{12 names}]
[, ROW_LABELS=*string_array*]
[, ROW_HEIGHTS=*array*]
[, SET_TABLE_SELECT=*[left, top, right, bottom]*]
[, SET_TABLE_VIEW=*[integer, integer]*]
[, SET_TEXT_SELECT=*[integer, integer]*]
[, SET_VALUE=*value*] [, TAB_MODE=*value*]
[, TABLE_BLANK=*cells*]
[, /TABLE_DISJOINT_SELECTION]
[, TABLE_XSIZE=*columns*] [, TABLE_YSIZE=*rows*]
[, /USE_TABLE_SELECT | , USE_TABLE_SELECT=*[left, top, right, bottom]*] [, /USE_TEXT_SELECT]

WIDGET_TEXT: [, /ALL_TEXT_EVENTS]
[, /APPEND] [, /CONTEXT_EVENTS] [, /EDITABLE]
[, GET_VALUE=*variable*]
[, IGNORE_ACCELERATORS={*string_array*|{0 | 1}}]
[, /INPUT_FOCUS] [, /KBRD_FOCUS_EVENTS]
[, /NO_NEWLINE] [, SET_TEXT_SELECT=*[integer, integer]*] [, SET_TEXT_TOP_LINE=*line_number*]
[, SET_VALUE=*value*] [, TAB_MODE=*value*]
[, /USE_TEXT_SELECT]

WIDGET_TREE: [, /CONTEXT_EVENTS]
[, SET_TREE_BITMAP=*array*]
[, /SET_TREE_EXPANDED] [, SET_TREE_SELECT={0 | 1 | *widget ID* | *array of widget IDs*}]
[, /SET_TREE_VISIBLE] [, TAB_MODE=*value*]

WIDGET_DISPLAYCONTEXTMENU - Displays a context-sensitive menu.

WIDGET_DISPLAYCONTEXTMENU, *Parent, X, Y, ContextBase_ID*

WIDGET_DRAW - Creates drawable widgets.

Result = WIDGET_DRAW(*Parent* [, /APP_SCROLL]
[, /BUTTON_EVENTS] [, CLASSNAME=*string*]
[, /COLOR_MODEL] [, COLORS=*integer*]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, /EXPOSE_EVENTS] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GRAPHICS_LEVEL=2]
[, GROUP_LEADER=*widget_id*]
[, IGNORE_ACCELERATORS=*value*]
[, KEYBOARD_EVENTS={0 | 1 | 2}]
[, KILL_NOTIFY=*string*] [, /MOTION_EVENTS]
[, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, RENDERER={0 | 1}]
[, RESOURCE_NAME=*string*] [, RETAIN={0 | 1 | 2}]
[, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*]
[, /SCROLL] [, /SENSITIVE] [, TOOLTIP=*string*]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, /VIEWPORT_EVENTS] [, /WHEEL_EVENTS]
[, XOFFSET=*value*] [, XSIZE=*value*]

[, X_SCROLL_SIZE=*width*] [, YOFFSET=*value*]
[, YSIZE=*value*] [, Y_SCROLL_SIZE=*height*])

WIDGET_DROPLIST - Creates droplist widgets.

Result = WIDGET_DROPLIST(*Parent*
[, /DYNAMIC_RESIZE] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, FONT=*string*]
[, FRAME=*value*] [, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, TAB_MODE=*value*] [, TITLE=*string*]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_EVENT - Returns events for the widget hierarchy.

Result = WIDGET_EVENT([*Widget_ID*])
[, BAD_ID=*variable*] [, /NOWAIT]
[, /SAVE_HOURGLASS]
UNIX Keywords: [, /YIELD_TO_TTY]

WIDGET_INFO - Obtains information about widgets.

Result = WIDGET_INFO([*Widget_ID*])
All widgets: [, /ACTIVE] [, /CHILD] [, /EVENT_FUNC]
[, /EVENT_PRO] [, FIND_BY_UNAME=string]
[, /FONTNAME] [, /GEOMETRY]
[, /KBRD_FOCUS_EVENTS] [, /MANAGED] [, /MAP]
[, /NAME] [, /PARENT] [, /PUSHBUTTON_EVENTS]
[, /REALIZED] [, /SENSITIVE] [, /SIBLING]
[, STRING_SIZE={*string* | [*string*, *font*]}]
[, /SYSTEM_COLORS] [, TAB_MODE=*value*]
[, /TRACKING_EVENTS] [, /TYPE] [, UNITS={0 | 1 | 2}] [, /UNAME] [, /UPDATE] [, /VALID_ID]
[, /VERSION] [, /VISIBLE]
WIDGET_BASE: [, /CONTEXT_EVENTS]
[, /MODAL] [, /TLB_ICONIFY_EVENTS]
[, /TLB_KILL_REQUEST_EVENTS]
[, /TLB_MOVE_EVENTS] [, /TLB_SIZE_EVENTS]
WIDGET_BUTTON: [, /BUTTON_SET]
[, /DYNAMIC_RESIZE] [, /PUSHBUTTON_EVENTS]
[, /TOOLTIP]
WIDGET_COMBOBOX: [, /COMBOBOX_GETTEXT]
[, /COMBOBOX_NUMBER] [, /DYNAMIC_RESIZE]
WIDGET_DRAW: [, /DRAW_BUTTON_EVENTS]
[, /DRAW_EXPOSE_EVENTS]
[, /DRAW_KEYBOARD_EVENTS]
[, /DRAW_MOTION_EVENTS]
[, /DRAW_VIEWPORT_EVENTS]
[, /DRAW_WHEEL_EVENTS] [, /TOOLTIP]
WIDGET_DROPLIST: [, /DROPLIST_NUMBER]
[, /DROPLIST_SELECT] [, /DYNAMIC_RESIZE]
WIDGET_LABEL: [, /DYNAMIC_RESIZE]
WIDGET_LIST: [, /CONTEXT_EVENTS]
[, /LIST_MULTIPLE] [, /LIST_NUMBER]
[, /LIST_NUM_VISIBLE] [, /LIST_SELECT]
[, /LIST_TOP]
WIDGET_PROPERTYSHEET:
[, /CONTEXT_EVENTS] [, COMPONENT=*objref*]
[, /MULTIPLE_PROPERTIES]
[, PROPERTY_VALID=*string*]
[, PROPERTY_VALUE=*string*]
[, /PROPERTYSHEET_NSELECTED]
[, /PROPERTYSHEET_SELECTED]
WIDGET_SLIDER: [, /SLIDER_MIN_MAX]
WIDGET_TAB: [, /TAB_CURRENT]
[, /TAB_MULTILINE] [, /TAB_NUMBER]
WIDGET_TABLE: [, /COLUMN_WIDTHS]
[, /CONTEXT_EVENTS] [, /ROW_HEIGHTS]
[, /STRING_SIZE] [, /TABLE_ALL_EVENTS]
[, /TABLE_BACKGROUND_COLOR]
[, /TABLE_DISJOINT_SELECTION]
[, /TABLE_EDITABLE] [, /TABLE_EDIT_CELL]
[, /TABLE_FONT]
[, /TABLE_FOREGROUND_COLOR]
[, /TABLE_SELECT] [, /TABLE_VIEW]
[, /USE_TABLE_SELECT]
WIDGET_TEXT: [, /CONTEXT_EVENTS]
[, /TEXT_ALL_EVENTS] [, /TEXT_EDITABLE]
[, /TEXT_NUMBER]
[, TEXT_OFFSET_TO_XY=*integer*]
[, /TEXT_SELECT] [, /TEXT_TOP_LINE]
[, TEXT_XY_TO_OFFSET=*[column, line]*]
WIDGET_TREE: [, /CONTEXT_EVENTS]
[, /TREE_EXPANDED] [, /TREE_ROOT]
[, /TREE_SELECT]

WIDGET_LABEL - Creates label widgets.

Result = WIDGET_LABEL(*Parent* [, /ALIGN_CENTER | , /ALIGN_LEFT | , /ALIGN_RIGHT]
[, /DYNAMIC_RESIZE] [, FONT=*string*]
[, FRAME=*width*] [, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, /SUNKEN_FRAME] [, /TRACKING_EVENTS]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, VALUE=*value*]
[, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_LIST - Creates list widgets.

Result = WIDGET_LIST(*Parent*
[, /CONTEXT_EVENTS] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, FONT=*string*]
[, FRAME=*width*] [, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /MULTIPLE]
[, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, TAB_MODE=*value*] [, /TRACKING_EVENTS]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, VALUE=*value*]
[, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_PROPERTYSHEET - Creates a property sheet widget, which exposes the properties of an IDL object. . This widget transparently handles property value changes

Result = WIDGET_PROPERTYSHEET(*Parent*
[, /ALIGN_BOTTOM |, /ALIGN_CENTER
|, /ALIGN_LEFT |, /ALIGN_RIGHT |, /ALIGN_TOP]
[, /CONTEXT_EVENTS] [, /EDITABLE]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FONT=STRING] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, IGNORE_ACCELERATORS=*value*]
[, KILL_NOTIFY=*string*]
[, /MULTIPLE_PROPERTIES] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, /SUNKEN_FRAME] [, /TRACKING_EVENTS]
[, UNAME=*string*] [,UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, VALUE=*value*]
[, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_SLIDER - Creates slider widgets.

Result = WIDGET_SLIDER(*Parent* [, /DRAG]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FONT=*string*] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, MAXIMUM=*value*]
[, MINIMUM=*value*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, SCROLL=*units*]
[, /SENSITIVE] [, /SUPPRESS_VALUE]
[, TAB_MODE=*value*] [, /TRACKING_EVENTS]
[, TITLE=*string*] [, UNAME=*string*] [, UNITS={0 | 1 |
2}] [, UVALUE=*value*] [, VALUE=*value*]
[, /VERTICAL] [, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_TAB - Creates tab widgets.

Result = WIDGET_TAB(*Parent* [, /ALIGN_BOTTOM | ,
/ALIGN_CENTER | , /ALIGN_LEFT | ,
/ALIGN_RIGHT | , /ALIGN_TOP]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, LOCATION={0 | 1 | 2 | 3}]
[, MULTILINE=0 | 1 (Windows) or *num* tabs per row
(Motif)] [, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, TAB_MODE=*value*] [, /TRACKING_EVENTS]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, XOFFSET=*value*]
[, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_TABLE - Creates table widgets.

Result = WIDGET_TABLE(*Parent* [, ALIGNMENT={0 |
1 | 2}] [, /ALL_EVENTS] [, AM_PM=*[string, string]*]
[, BACKGROUND_COLOR=*array*]
[, COLUMN_LABELS=*string_array*]
[, /COLUMN_MAJOR | , /ROW_MAJOR]
[, COLUMN_WIDTHS=*array*]
[, /CONTEXT_EVENTS]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, /DISJOINT_SELECTION] [, /EDITABLE]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FONT=*string*] [, FOREGROUND_COLOR=*array*]
[, FORMAT=*value*] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, IGNORE_ACCELERATORS=*value*]
[, /KBRD_FOCUS_EVENTS] [, KILL_NOTIFY=*string*]
[, MONTHS=*string_array*{12 names}]
[, /NO_COLUMN_HEADERS] [, /NO_COPY]
[, /NO_HEADERS] [, /NO_ROW_HEADERS]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, /RESIZEABLE_COLUMNS]
[, /RESIZEABLE_ROWS{not supported in Windows}]
[, RESOURCE_NAME=*string*]
[, ROW_HEIGHTS=*array*]
[, ROW_LABELS=*string_array*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SCROLL] [, /SENSITIVE]
[, TAB_MODE=*value*] [, /TRACKING_EVENTS]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, VALUE=*value*]
[, XOFFSET=*value*] [, XSIZE=*value*]
[, X_SCROLL_SIZE=*width*] [, YOFFSET=*value*]
[, YSIZE=*value*] [, Y_SCROLL_SIZE=*height*])

WIDGET_TEXT - Creates text widgets.

Result = WIDGET_TEXT(*Parent* [, /ALL_EVENTS] [, /CONTEXT_EVENTS] [, /EDITABLE] [, EVENT_FUNC=*string*] [, EVENT_PRO=*string*] [, FONT=*string*] [, FRAME=*width*] [, FUNC_GET_VALUE=*string*] [, GROUP_LEADER=*widget_id*] [, IGNORE_ACCELERATORS=*value*] [, /KBRD_FOCUS_EVENTS] [, KILL_NOTIFY=*string*] [, /NO_COPY] [, /NO_NEWLINE] [, NOTIFY_REALIZE=*string*] [, PRO_SET_VALUE=*string*] [, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*] [, /SCROLL] [, /SENSITIVE] [, TAB_MODE=*value*] [, /TRACKING_EVENTS] [, UNAME=*string*] [, UNITS={0 | 1 | 2}] [, UVALUE=*value*] [, VALUE=*value*] [, /WRAP] [, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_TREE - Creates tree widgets.

Result = WIDGET_TREE(*Parent* [, /ALIGN_BOTTOM | , /ALIGN_CENTER | , /ALIGN_LEFT | , /ALIGN_RIGHT | , /ALIGN_TOP] [, BITMAP=*array*] [, /CONTEXT_EVENTS] [, EVENT_FUNC=*string*] [, EVENT_PRO=*string*] [, ./EXPANDED] [, /FOLDER] [, FUNC_GET_VALUE=*string*] [, GROUP_LEADER=*widget_id*] [, KILL_NOTIFY=*string*] [, /MULTIPLE] [, /NO_COPY] [, NOTIFY_REALIZE=*string*] [, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*] [, /SENSITIVE] [, TAB_MODE=*value*] [, /TRACKING_EVENTS] [, /TOP] [, UNAME=*string*] [, UNITS={0 | 1 | 2}] [, UVALUE=*value*] [, VALUE=*string*] [, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])

WINDOW - Creates window for the display of graphics or text.

WINDOW [, *Window_Index*] [, COLORS=*value*] [, /FREE] [, /PIXMAP] [, RETAIN={0 | 1 | 2}] [, TITLE=*string*] [, XPOS=*value*] [, YPOS=*value*] [, XSIZE=*pixels*] [, YSIZE=*pixels*]

WRITE_BMP - Writes Microsoft Windows Version 3 device independent bitmap file (.BMP).

WRITE_BMP, *Filename*, *Image*[, *R*, *G*, *B*] [, /FOUR_BIT] [, IHDR=*structure*] [, HEADER_DEFINE=*h*{define *h* before call}] [, /RGB]

WRITE_GIF - Writes a Graphics Interchange Format (GIF) file.

WRITE_GIF, *Filename*, *Image*[, *R*, *G*, *B*] [, /MULTIPLE [, /CLOSE]]

WRITE_IMAGE - Writes an image and its color table vectors, if any, to a file of a specified type.

WRITE_IMAGE, *Filename*, *Format*, *Data* [, *Red*, *Green*, *Blue*] [, /APPEND]

WRITE_JPEG - Writes a JPEG file.

WRITE_JPEG [, *Filename* | , UNIT=*lun*] , *Image* [, /ORDER] [, /PROGRESSIVE] [, QUALITY=*value*{0 to 100}] [, TRUE={1 | 2 | 3}]

WRITE_JPEG2000 - Writes a JPEG2000 file.

WRITE_JPEG2000, *Filename*, *Image* [, Red, Green, Blue] [, N_LAYERS=*value*] [, N_LEVELS=*value*] [, /ORDER] [, /REVERSIBLE])

WRITE_NRIF - Writes NCAR Raster Interchange Format rasterfile.

WRITE_NRIF, *File*, *Image* [, *R*, *G*, *B*]

WRITE_PICT - Writes Macintosh PICT (version 2) bitmap file.

WRITE_PICT, *Filename* [, *Image*, *R*, *G*, *B*]

WRITE_PNG - Writes Portable Network Graphics (PNG) file.

WRITE_PNG, *Filename*, *Image*[, *R*, *G*, *B*] [, /VERBOSE] [, TRANSPARENT=*array*] [, /ORDER]

WRITE_PPM - Writes PPM (true-color) or PGM (gray scale) file.

WRITE_PPM, *Filename*, *Image* [, /ASCII]

WRITE_SPR - Writes row-indexed sparse array structure to a file.

WRITE_SPR, *AS*, *Filename*

WRITE_SRF - Writes Sun Raster File (SRF).

WRITE_SRF, *Filename* [, *Image*, *R*, *G*, *B*] [, /ORDER] [, /WRITE_32]

WRITE_SYLK - Writes SYLK (Symbolic Link) spreadsheet file.

Result = WRITE_SYLK(*File*, *Data* [, STARTCOL=*column*] [, STARTROW=*row*])

WRITE_TIFF - Writes TIFF file with 1 to 3 channels.

WRITE_TIFF, *Filename* [, *Image*] [, /APPEND] [, BITS_PER_SAMPLE={1 | 4 | 8}] [, RED=*value*] [, GREEN=*value*] [, BLUE=*value*] [, /CMYK] [, COMPRESSION={0 | 1 | 2 | 3}] [, DESCRIPTION=*string*] [, DOCUMENT_NAME=*string*] [, DOT_RANGE=*intarray*] [, GEOTIFF=*structure*] [, ICC_PROFILE=*array*] [, /LONG | , /SHORT | ,/FLOAT] [, ORIENTATION=*value*] [, PHOTOSHOP=*bytearray*] [, PLANARCONFIG={1 | 2}] [, /VERBOSE] [, XPOSITION=*units*] [, XRESOL=*pixels/inch*] [, YPOSITION=*units*] [, YRESOL=*pixels/inch*]

Note: LZW compression (COMPRESSION=1) is only available with the appropriate license feature.

WRITE_WAV - Writes the audio stream to the named .WAV file.

WRITE_WAV, *Filename* , *Data* [, *Rate*]

WRITE_WAVE - Writes Wavefront Advanced Visualizer (.WAV) file.

WRITE_WAVE, *File*, *Array* [, /BIN]
[, DATANAME=*string*] [, MESHNAME=*string*]
[, /NOMESHDEF] [, /VECTOR]

WRITEU - Writes unformatted binary data to a file.

WRITEU, *Unit*, $Expr_1$ ..., $Expr_n$
[, TRANSFER_COUNT=*variable*]

WSET - Selects the current window.

WSET [, *Window_Index*]

WSHOW - Exposes or hides the designated window.

WSHOW [, *Window_Index* [, *Show*]] [, /ICONIC]

WTN - Returns wavelet transform of the input array.

Result = WTN(*A*, *Coef* [, /COLUMN] [, /DOUBLE]
[, /INVERSE] [, /OVERWRITE])

X

XBM_EDIT - Creates, edits bitmap icons for IDL widget button labels.

XBM_EDIT [, /BLOCK] [, FILENAME=*string*]
[, GROUP=*widget_id*] [, XSIZE=*pixels*]
[, YSIZE=*pixels*]

XDISPLAYFILE - Displays ASCII text file in scrolling text widget.

XDISPLAYFILE, *Filename* [, /BLOCK]
[, DONE_BUTTON=*string*] [, /EDITABLE]
[, FONT=*string*] [, GROUP=*widget_id*]
[, /GROW_TO_SCREEN] [, HEIGHT=*lines*]
[, /MODAL] [, RETURN_ID=*variable*] [, TEXT=*string or string array*] [, TITLE=*string*] [, WIDTH=*characters*]
[, WTEXT=*variable*]

XDXF - Utility for displaying and interactively manipulating DXF objects

XDXF [, *Filename*] [, /BLOCK] [, GROUP=*widget_id*]
[, SCALE=*value*] [, /TEST] [*keywords to XOBJVIEW*]

XFONT - Creates modal widget to select and view an X Windows font.

Result = XFONT([, GROUP=*widget_id*]
[, /PRESERVE_FONT_INFO])

XINTERANIMATE - Displays animated sequence of images.

XINTERANIMATE [, *Rate*]
Keywords for initialization: [, SET=*[sizex, sizey, nframes]*] [, /BLOCK] [, /CYCLE]
[, GROUP=*widget_id*] [, /MODAL] [, MPEG_BITRATE=*value*]
[, MPEG_IFRAME_GAP=*integer value*]
[, MPEG_MOTION_VEC_LENGTH={1 | 2 | 3}]
[, /MPEG_OPEN, MPEG_FILENAME=*string*]
[MPEG_QUALITY=*value*{0 to 100}] [, /SHOWLOAD]
[, /TRACK] [, TITLE=*string*]
Keywords for loading images: [, FRAME=*value*{0 to (*nframes*-1)}[, IMAGE=*value*]] [, /ORDER]
[, WINDOW=*[window_num* [, *x0, y0, sx, sy*]*]*]
Keywords for running animations: [, /CLOSE]
[, /KEEP_PIXMAPS] [, /MPEG_CLOSE]
[, XOFFSET=*pixels*] [, YOFFSET=*pixels*]

XLOADCT - Provides GUI to interactively select and load color tables.

XLOADCT [, /BLOCK] [, BOTTOM=*value*]
[, FILE=*string*] [, GROUP=*widget_id*] [, /MODAL]
[, NCOLORS=*value*] [, /SILENT]
[, UPDATECALLBACK='*procedure_name*'
[, UPDATECBDATA=*value*]] [, /USE_CURENT]

XMANAGER - Provides event loop manager for IDL widgets.

XMANAGER [, *Name*, *ID*] [, /CATCH]
[, CLEANUP=*string*]
[, EVENT_HANDLER=*procedure*]
[, GROUP_LEADER=*widget_id*] [, /JUST_REG]
[, /NO_BLOCK]

XMNG_TMPL - Template for creating widgets.

XMNG_TMPL [, /BLOCK] [, GROUP=*widget_id*]

XMTOOL - Displays tool for viewing XMANAGER widgets.

XMTOOL [, /BLOCK] [, GROUP=*widget_id*]

XOBJVIEW - Displays object viewer widget.

XOBJVIEW, *Obj* [, BACKGROUND=*[r, g, b]*]
[, /BLOCK] [, /DOUBLE_VIEW] [, GROUP=*widget_id*]
[, .JUST_REG] [, /MODAL] [, REFRESH=*widget_id*]
[, RENDERER={0|1}] [, SCALE=*value*]
[, STATIONARY=*objref(s)*] [, /TEST] [, TITLE=*string*]
[, TLB=*variable*] [, XOFFSET=*value*] [, XSIZE=*pixels*]
[, YOFFSET=*value*] [, YSIZE=*pixels*]

XOBJVIEW_ROTATE - Programmatically rotate the object currently displayed in XOBJVIEW.

XOBJVIEW_ROTATE, *Axis*, *Angle* [, /PREMULTIPLY]

XOBJVIEW_WRITE_IMAGE - Write the object currently displayed in XOBJVIEW to an image file.

XOBJVIEW_WRITE_IMAGE, *Filename*, *Format*
[, DIMENSIONS=*[x, y]*]

XPALETTE - Displays widget used to create and modify color tables.

XPALETTE [, /BLOCK] [, GROUP=*widget_id*]
[, UPDATECALLBACK='*procedure_name*'
[, UPDATECBDATA=*value*]]

XPCOLOR - Adjusts the value of the current foreground plotting color, !P.COLOR.

XPCOLOR [, GROUP=*widget_id*]

XPLOT3D - Utility for creating and interactively manipulating 3D plots.

XPLOT3D, *X*, *Y*, *Z* [, /BLOCK] [, COLOR=*[r,g,b]*] [, /DOUBLE_VIEW] [, GROUP=*widget_id*] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]] [, /MODAL] [, NAME=*string*] [, /OVERPLOT] [, SYMBOL=*objref(s)*] [, /TEST] [, THICK=*points*{1.0 to 10.0}] [, TITLE=*string*] [, XRANGE=*[min, max]*] [, YRANGE=*[min, max]*] [, ZRANGE=*[min, max]*] [, XTITLE=*string*] [, YTITLE=*string*] [, ZTITLE=*string*]

XREGISTERED - Returns registration status of a given widget.

Result = XREGISTERED(*Name* [, /NOSHOW])

XROI - Utility for interactively creating and obtaining information about ROIs.

XROI [, *ImageData*] [, *R*] [, *G*] [, *B*] [, /BLOCK] [[, /FLOATING] , GROUP=*widget_ID*] [, /MODAL] [, REGIONS_IN=*value*] [, REGIONS_OUT=*value*] [, REJECTED=*variable*] [, RENDERER={0 | 1}] [, ROI_COLOR=*[r, g, b] or variable*] [, ROI_GEOMETRY=*variable*] [, ROI_SELECT_COLOR=*[r, g, b] or variable*] [, STATISTICS=*variable*] [, TITLE=*string*] [, TOOLS=*string or string array*{valid values are 'Freehand Draw', 'Polygon Draw', and 'Selection'}] [, X_SCROLL_SIZE=*vlaue*] [, Y_SCROLL_SIZE=*value*]

XSQ_TEST - Computes Chi-square goodness-of-fit test.

Result = XSQ_TEST(*Obfreq*, *Exfreq* [, EXCELL=*variable*] [, OBCELL=*variable*] [, RESIDUAL=*variable*])

XSURFACE - Provides GUI to SURFACE and SHADE_SURF.

XSURFACE, *Data* [, /BLOCK] [, GROUP=*widget_id*]

XVAREDIT - Provides widget-based editor for IDL variables.

XVAREDIT, *Var* [, NAME='*variable_name*'{ignored if variable is a structure}] [, GROUP=*widget_id*] [, X_SCROLL_SIZE=*columns*] [, Y_SCROLL_SIZE=*rows*]

XVOLUME - Utility for viewing and interactively manipulating volumes and isosurfaces.

XVOLUME, *Vol*, [, /BLOCK] [, GROUP=*widget_id*] [, /INTERPOLATE] [, /MODAL] [, RENDERER={0 | 1}] [, /REPLACE] [, SCALE=*value*] [, /TEST] [, XSIZE=*pixels*] [, YSIZE=*pixels*]

XYOUTS - Draws text on currently-selected graphics device.

XYOUTS, [*X*, *Y*,] *String* [, ALIGNMENT=*value*{0.0 to 1.0}] [, CHARSIZE=*value*] [, CHARTHICK=*value*] [, TEXT_AXES={0 | 1 | 2 | 3 | 4 | 5}] [, WIDTH=*variable*]

Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, ORIENTATION=*ccw_degrees_from_horiz*] [, /NOCLIP] [, /T3D] [, Z=*value*]

Z

ZOOM - Zooms portions of the display.

ZOOM [, /CONTINUOUS] [, FACT=*integer*] [, /INTERP] [, /KEEP] [, /NEW_WINDOW] [, XSIZE=*value*] [, YSIZE=*value*] [, ZOOM_WINDOW=*variable*]

ZOOM_24 - Zooms portions of true-color (24-bit) display.

ZOOM_24 [, FACT=*integer*] [, /RIGHT] [, XSIZE=*value*] [, YSIZE=*value*]

Objects

This section lists all IDL objects and their methods. In addition to the syntax conventions discussed in "IDL Syntax Conventions" on page 28, note the following:

- The *Object_Name*::Init method for each object has keywords that are followed by either {Get}, {Set}, or {Get, Set}. Properties retrievable via *Object_Name*::GetProperty are indicated by {Get}; properties settable via *Object_Name*::SetProperty are indicated by {Set}. Properties that are both retrievable and settable are indicated by {Get, Set}. Do not include the braces, Get, or Set in your call.
- Each object's Cleanup method lists two possible syntaxes. The second syntax (*Obj->Object_Name*::Cleanup) can be used only in a subclass' Cleanup method.
- Some objects have Init methods that list two possible syntaxes. The second syntax (*Obj->Object_Name*::Init) can be used only in a subclass' Init method.

IDL_Container - Object used to hold other objects. No superclasses. Subclasses: IDLgrModel IDLgrScene IDLgrView IDLgrViewgroup.

IDL_Container::Add - Adds a child object to the container.
Obj->[IDL_Container::]Add,*Object* [POSITION=*index*]

IDL_Container::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDL_Container::]Cleanup

IDL_Container::Count - Returns the number of objects contained by the container object.
Result = *Obj*->[IDL_Container::]Count( )

IDL_Container::Get - Returns an array of object references to objects in a container.
Result = *Obj*->[IDL_Container::]Get(
[, /ALL [, ISA=*class_name(s)*] | , POSITION=*index*]
[COUNT=*variable*])

IDL_Container::Init - Initializes the container object.
Obj = OBJ_NEW('IDL_Container')
Result = *Obj*->[IDL_Container::]Init( )

IDL_Container::IsContained - Returns true (1) if the specified object is in the container, or false (0) otherwise.
Result = *Obj*->[IDL_Container::]IsContained(*Object*
[, POSITION=*variable*])

IDL_Container::Move - Moves an object from one position in a container to a new position.
Obj->[IDL_Container::]Move, *Source*, *Destination*

IDL_Container::Remove - Removes an object from the container.
Obj->[IDL_Container::]Remove [, *Child_object* | ,
POSITION=*index* | , /ALL]

IDL_Savefile - Object that provides complete query and restore capabilities for IDL SAVE files.
Properties: None.

IDL_Savefile::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDL_Savefile::]Cleanup

IDL_Savefile::Contents - returns a structure variable of type IDL_SAVEFILE_CONTENTS containing information about the associated SAVE file and its contents.
Result = *Obj*->[IDL_Savefile::]Contents()

IDL_Savefile::Init - Initializes the Safefile object.
Obj = OBJ_NEW('IDL_Savefile' [, *Filename*]
[, FILENAME=*string*]
[, /RELAXED_STRUCTURE_ASSIGNMENT])
Result = *Obj*->[IDL_Savefile::]Init([, *Filename*]
[, FILENAME=*string*]
[, /RELAXED_STRUCTURE_ASSIGNMENT])

IDL_Savefile::Names - Returns names, or heap variable identifiers, of items contained within the SAVE file.
Result = *Obj*->[IDL_SAVEFILE::]Names(
[, COUNT=*variable*] [, /COMMON_BLOCK |
, COMMON_VARIABLE=*string* | , /FUNCTION |
, /OBJECT_HEAPVAR | , /POINTER_HEAPVAR |
, /PROCEDURE | , /STRUCTURE_DEFINITION |
, /SYSTEM_VARIABLE])

IDL_Savefile::Restore - Selectively restores individual items from the associated SAVE file.
Obj->[IDL_Savefile::]Restore, *SaveItem* [, /COMMON]
[, /FUNCTION] [, NEW_HEAPVAR=*variable*]
[, /OBJECT_HEAPVAR] [, /POINTER_HEAPVAR]
[, /PROCEDURE] [, RESTORED_OBJECTS=*variable*]
[, /STRUCTURE_DEFINITION] [, /VERBOSE]

IDL_Savefile::Size - Returns the size and type information for the specified variable, system variable, or heap variable in the SAVE file .
Result = *Obj*->[IDL_Savefile::]Size(*SaveItem*)

IDLanROI - Represents a region of interest. Superclass of IDLgrROI.

Properties: [, ALL{Get}=*variable*] [, BLOCKSIZE{Get, Init, Set}=*vertices*] [, DATA{Get, Init, Set}=*array*] [, DOUBLE{Get, Init, Set}=*value*] [, /INTERIOR{Get, Init, Set}] [, N_VERTS{Get}=*variable*] [, ROI_XRANGE{Get}=*variable*] [, ROI_YRANGE{Get}=*variable*] [, ROI_ZRANGE{Get}=*variable*] [, TYPE{Get, Init}={ 0 | 1 | 2 }]

IDLanROI::AppendData - Appends vertices to the region.

Obj–>[IDLanROI::]AppendData, *X* [, *Y*] [, *Z*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

IDLanROI::Cleanup - Performs all cleanup for the object.

Obj–>[IDLanROI::]Cleanup or OBJ_DESTROY, *Obj*

IDLanROI::ComputeGeometry - Computes the geometrical values for area, perimeter, and/or centroid of the region.

Result = *Obj*–>[IDLanROI::]ComputeGeometry([, AREA=*variable*] [, CENTROID=*variable*] [, PERIMETER=*variable*] [, SPATIAL_OFFSET=*vector*] [, SPATIAL_SCALE=*vector*])

IDLanROI::ComputeMask - Prepares a two-dimensional mask for the region.

Result = *Obj*–>[IDLanROI::]ComputeMask([, INITIALIZE={ –1 | 0 | 1 }] [, DIMENSIONS=*[xdim, ydim]*] | [, MASK_IN=*array*] [, LOCATION=*[x, y* [, *z]]*] [, MASK_RULE={ 0 | 1 | 2 }] [, PIXEL_CENTER=[*x, y*]] [, PLANE_NORMAL=*[x, y, z]*] [, PLANE_XAXIS=*[x,y,z]*] [, RUN_LENGTH=*value*])

IDLanROI::ContainsPoints - Determines whether the given data coordinates are contained within the closed polygon region.

Result = *Obj*–>[IDLanROI::]ContainsPoints(*X* [, *Y* [, *Z*]])

IDLanROI::GetProperty - Retrieves the value of a property or group of properties for the region.

Obj–>[IDLanROI::]GetProperty[, *PROPERTY*=*value*]

IDLanROI::Init - Initializes a region of interest object.

Obj = OBJ_NEW('IDLanROI' [, *X* [, *Y* [, *Z*]]] [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLanROI::]Init([*X* [, *Y* [, *Z*]]] [, *PROPERTY*=*value*])

IDLanROI::RemoveData - Removes vertices from the region.

Obj–>[IDLanROI::]RemoveData[, COUNT=*vertices*] [, START=*index*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

IDLanROI::ReplaceData - Replaces vertices in the region with alternate values.

Obj–>[IDLanROI::]ReplaceData, *X*[, *Y*[, *Z*]] [, START=*index*] [, FINISH=*index*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

IDLanROI::Rotate - Modifies the vertices for the region by applying a rotation.

Obj–>[IDLanROI::]Rotate, *Axis*, *Angle* [, CENTER=*[x, y*[, *z]]*]

IDLanROI::Scale - Modifies the vertices for the region by applying a scale.

Obj–>[IDLanROI::]Scale, *Sx*[, *Sy*[, *Sz*]]

IDLanROI::SetProperty - Sets the value of a property or group of properties for the region.

Obj–>[IDLanROI::]SetProperty[, *PROPERTY*=*value*]

IDLanROI::Translate - Modifies the vertices for the region by applying a translation.

Obj–>[IDLanROI::]Translate, *Tx*[, *Ty*[, *Tz*]]

IDLanROIGroup - This object is an analytical representation of a group of regions of interest. Subclass of IDL_Container. Superclass of IDLgrROIGroup.

Properties: [, ALL{Get}=*variable*] [, ROIGROUP_XRANGE{Get}=*variable*] [, ROIGROUP_YRANGE{Get}=*variable*] [, ROIGROUP_ZRANGE{Get}=*variable*]

IDLanROIGroup::Add - Adds a region to the region group.

Obj–>[IDLanROIGroup::]Add, *ROI*

IDLanROIGroup::Cleanup - Performs all cleanup for the object.

OBJ_DESTROY, *Obj*

or *Obj*–>[IDLanROIGroup::]Cleanup

IDLanROIGroup::ContainsPoints - Determines whether the given points (in data coordinates) are contained within the closed polygon regions within this group.

Result = *Obj*–>[IDLanROIGroup::]ContainsPoints(*X*[, *Y*[, *Z*]])

IDLanROIGroup::ComputeMask - Prepares a 2-D mask for this group of regions.

Result = *Obj*–>[IDLanROIGroup::]ComputeMask([, INITIALIZE={ –1 | 0 | 1 }] [, DIMENSIONS=*[xdim, ydim]*] | [, MASK_IN=*array*] [, LOCATION=*[x, y* [, *z]]*] [, MASK_RULE={ 0 | 1 | 2 }] [, RUN_LENGTH=*value*])

IDLanROIGroup::ComputeMesh - Triangulates a surface mesh with optional capping from the stack of regions contained within this group.

Result = *Obj*->[IDLanROIGroup::]ComputeMesh(*Vertices*, *Conn* [, CAPPED={ 0 | 1 | 2}] [, SURFACE_AREA=*variable*])

IDLanROIGroup::GetProperty - Retrieves the value of a property or group of properties for the region group.

Obj–>[IDLanROIGroup::]GetProperty [, *PROPERTY=variable*]

IDLanROIGroup::Init - Initializes a region of interest group object.

Obj = OBJ_NEW('IDLanROIGroup') or *Result* = *Obj*–>[IDLanROIGroup::]Init( )

IDLanROIGroup::Rotate - Modifies the vertices for all regions within the group by applying a rotation.

Obj–>[IDLanROIGroup::]Rotate, *Axis*, *Angle*[, CENTER=*[x, y*[, *z*]*]*]

IDLanROIGroup::Scale - Modifies the vertices for the region by applying a scale.

Obj–>[IDLanROIGroup::]Scale, *Sx*[, *Sy*[, *Sz*]]

IDLanROIGroup::Translate - Modifies the vertices of all regions within the group by applying a translation.

Obj–>[IDLanROIGroup::]Translate, *Tx*[, *Ty*[, *Tz*]]

IDLcomActiveX - Creates an IDL object that encapsulates an ActiveX control.

IDLcomIDispatch - Creates a COM object that implements an IDispatch interface. A dynamic sub-class of IDLcomIDispatch is created when the object is instantiated.

IDLcomIDispatch::GetProperty - Get properties for an IDispatch interface.

IDLcomIDispatch->GetProperty, *PROPERTY=variable*, [*arg0*, *arg1*, …]

IDLcomIDispatch::Init - Initialize a COM object and establish a link between the resulting IDL object and the IDispatch interface

Obj = OBJ_NEW('IDLcomIDispatch$*IDTYPE*$ID')

IDLcomIDispatch::SetProperty - Set properties for an IDispatch interface.

IDLcomIDispatch->SetProperty, *PROPERTY=value*

IDLffDICOM - Contains the data for one or more images embedded in a DICOM part 10 file. No superclasses. No subclasses.

Property: [, /VERBOSE{Init}]

IDLffDICOM::Cleanup - Destroys the IDLffDICOM object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLffDICOM::]Cleanup

IDLffDICOM::DumpElements - Dumps a description of the DICOM data elements of IDLffDICOM object to the screen or to a file.

Obj->[IDLffDICOM::]DumpElements [, *Filename*]

IDLffDICOM::GetChildren - Finds the member element references of a DICOM sequence.

array = *Obj*->[IDLffDICOM::]GetChildren(*Reference*)

IDLffDICOM::GetDescription - Takes optional DICOM group and element arguments and returns array of STRING descriptions.

array = *Obj*->[IDLffDICOM::]GetDescription([*Group* [, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetElement - Takes optional DICOM group and/or element arguments and returns an array of DICOM Element numbers for those parameters.

array = *Obj*->[IDLffDICOM]GetElement([*Group* [, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetGroup - Takes optional DICOM group and/or element arguments and returns an array of DICOM Group numbers for those parameters.

array = *Obj*->[IDLffDICOM::]GetGroup([*Group* [, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetLength - Takes optional DICOM group and/or element arguments and returns an array of LONGs.

array = *Obj*->[IDLffDICOM::]GetLength([*Group* [, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetParent - Finds the parent references of a set of elements in a DICOM sequence.

array = *Obj*->[IDLffDICOM::]GetParent(*ReferenceList*)

IDLffDICOM::GetPreamble - Returns the preamble of a DICOM v3.0 Part 10 file.

array = *Obj*->[IDLffDICOM::]GetPreamble( )

IDLffDICOM::GetReference - Takes optional DICOM group and/or element arguments and returns an array of references to matching elements in the object.

array = *Obj*->[IDLffDICOM::]GetReference([*Group* [, *Element*]] [, DESCRIPTION=*string*] [, VR=*DICOM VR string*])

IDLffDICOM::GetValue - Takes optional DICOM group and/or element arguments and returns an array of POINTERs to the values of the elements matching those parameters.

ptrArray = *Obj*->[IDLffDICOM::]GetValue([*Group* [, *Element*]] [, REFERENCE=*list of element references*] [, /NO_COPY])

IDLffDICOM::GetVR - Takes optional DICOM group and/or element arguments and returns an array of VR (Value Representation) STRINGs for those parameters.

array = *Obj*->[IDLffDICOM::]GetVR([*Group* [, *Element*]] [, REFERENCE=*list of references*])

IDLffDICOM::Init - Creates a new IDLffDICOM object and optionally reads the specified file as defined in the IDLffDICOM::Read method.

Result = OBJ_NEW('IDLffDICOM' [, *Filename*] [, *PROPERTY=value*]) or

Result = *Obj*->[IDLffDICOM::]Init([, *Filename*] [, *PROPERTY=value*])

IDLffDICOM::Read - Opens and reads from the specified disk file, places the information into the DICOM object, then closes the file.

result = *Obj*->[IDLffDICOM::]Read(*Filename* [, ENDIAN={1 | 2 | 3 | 4}])

IDLffDICOM::Reset - Removes all of the elements from the IDLffDICOM object, leaving the object otherwise intact.

Obj->[IDLffDICOM::]Reset

IDLffDicomEx Object - See Appendix B, "IDL DICOM Quick Reference" in the *Medical Imaging in IDL* manual.

IDLffDXF - Object that contains geometry, connectivity, and attributes for graphics primitives. No superclasses. No subclasses.

IDLffDXF::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLffDXF::]Cleanup

IDLffDXF::GetContents - Returns the DXF entity types contained in the object.

Result = *Obj*->[IDLffDXF::]GetContents([*Filter*] [BLOCK=*string*] [, COUNT=*variable*] [LAYER=*string*])

IDLffDXF::GetEntity - Returns an array of vertex data for the requested entity type.

Result = *Obj*->[IDLffDXF::]GetEntity(*Type* [, BLOCK=*string*] [, INDEX=*value*] [, LAYER=*string*])

IDLffDXF::GetPalette - Returns current color table in the object.

Obj->[IDLffDXF::]GetPalette, *Red*, *Green*, *Blue*

IDLffDXF::Init - Initializes the DXF object.

Result = OBJ_NEW('IDLffDXF' [, *Filename*])

or

Result = *Obj*->[IDLffDXF::]Init([*Filename*])

IDLffDXF::PutEntity - Inserts an entity into the DXF object.

Obj->[IDLffDXF::]PutEntity, *Data*

IDLffDXF::Read - Reads a file, parsing the DXF object information contained in the file, and inserts it into itself.

Result = *Obj*->[IDLffDXF::]Read(*Filename*)

IDLffDXF::RemoveEntity - Removes the specified entity or entities from the DXF object.

Obj->[IDLffDXF::]RemoveEntity[, *Type*] [, INDEX=*value*]

IDLffDXF::Reset - Removes all the entities from the DXF object.

Obj->[IDLffDXF::]Reset

IDLffDXF::SetPalette - Sets the current color table in the object.

Obj->[IDLffDXF::]SetPalette, *Red*, *Green*, *Blue*

IDLffDXF::Write - Writes a file for the DXF entity information this object contains.

Result = *Obj*->[IDLffDXF::]Write(*Filename*)

IDLffJPEG2000 - Object class used for reading and writing JPEG2000 files. No superclasses. No subclasses

Properties: [, BIT_DEPTH {Get, Init, Set}=*integer vector*] [, BIT_RATE {Init}=*array*] [, COLOR_SPACE {Get}=*variable*] [, COMMENT {Get, Init, Set}=*string vector*] [, DIMENSIONS {Get, Init, Set}=*vector*] [, DISPLAY_RESOLUTION {Get, Init, Set}=*vector*] [, FILENAME {Get, Init,}=*string*] [, JP2 {Get}=0 | 1] [, N_COMPONENTS {Get, Init, Set}=*integer*] [, N_LAYERS {Get, Init, Set}=*integer*] [, N_LEVELS {Get, Init, Set}=*integer*] [, N_TILES {Get}=*variable*] [, OFFSET {Get, Init, Set}=*integer*] [, PALETTE {Get, Init, Set}=*array*] [, PERSISTENT {Init,}=0 | 1] [, PROGRESSION {Get, Init, Set}=*string*] [, QUIET {Init, Set}=0 | 1] [, READ {Get, Init}=0 | 1] [, REVERSIBLE {Get, Init, Set}=0 | 1] [, SIGNED {Get, Init, Set}=0 | 1] [, SUBSAMPLING {Get}=*array*] [, TILE_DIMENSIONS {Get, Init, Set}=*vector*] [, TILE_OFFSET {Get, Init, Set}=*vector*] [, TILE_RANGE {Get, Init, Set}=*vector*] [, UUIDS {Get}=*array*] [, WRITE {Get, Init,}=0 | 1] [, XML {Get, Init, Set}=*string vector*] [, YCC {Get, Init, Set}=0 | 1]

IDLffJPEG2000::GetData - Returns image data from the IDLffJPEG2000 object.

Result = Obj->[IDLffJPEG2000::]GetData([COMPONENT=*value*] [, DISCARD_LEVELS=*value*] [, MAX_LAYERS=*value*] [, N_COMPONENTS=*value*] [, ORDER=*value*] [, REGION=*value*] [, /RGB] [, TILE_INDEX=*value*])

IDLffJPEG2000::GetProperty - Retrieves the value of a property or group of properties for the IDLffJPEG2000 object.

Obj->[IDLgrJPEG2000::]GetProperty [, *PROPERTY*=*variable*]

IDLffJPEG2000::GetTileProperty - Retrieves the properties of a tile in an IDLffJPEG2000 object.

Obj->[IDLffJPEG2000:]GetTileProperty [, *TileIndex* [, *TileComponent*]] [, N_LAYERS=*variable*] [, N_LEVELS=*variable*] [, PROGRESSION=*variable*] [, REVERSIBLE=*variable*] [, TILE_DIMENSIONS=*variable*] [, TILE_OFFSET=*variable*] [, YCC=*variable*]

IDLffJPEG2000::GetUUID - Allows you to get the data field from the specified UUID box.

Data = *Obj*-> [IDLffJPEG2000::]GetUUID(UUID, LENGTH=*length*)

IDLffJPEG2000::Init - Initializes an IDLffJPEG2000 object.

Obj = OBJ_NEW('IDLffJPEG2000' , *Filename*[, PROPERTY=*value*])

IDLffJPEG2000::SetData - Writes data to the IDLffJPEG2000 object.

Obj->[IDLffJPEG2000::]SetData ([*P1*, ..., *Pn*] [, COMPONENT=*value*] [, /ORDER] [, TILE_INDEX=*value*]

IDLffJPEG2000::SetProperty - Sets the properties of an object that are open for writing.

Obj->[IDLffJPEG2000:]SetProperty [, PROPERTY=*value*]

IDLffJPEG2000::SetUUID - Allows you to add UUID boxes when creating a new JPEG2000 file.

Obj->[IDLffJPEG2000:]SetUUID, uuid, Data

IDLffLangCat - Object class is used to find and load an XML language catalog. No superclasses. No subclasses

Properties: [, APP_NAME {Get, Init}=*string*] [, APP_PATH {Get, Init}=*string*] [, AVAILABLE_LANGUAGES {Get}=*string*] [, DEFAULT_KEYS {Get}=*string*] [, DEFAULT_LANGUAGE {Get, Init}=*string*] [, DEFAULT_N_KEYS {Get}=*variable*] [, FILENAME {Get, Init}=*string*] [, KEYS {Get}=*variable*] [, LANGUAGE {Get, Init, Set}=*string*] [, N_KEYS {Get}=*variable*] [, VERBOSE {Get, Init, Set}=*byte*]

IDLffLangCat::AppendCatalog - Adds keys from a file or files to those used to build the language catalog.

Result = Obj->[IDLffLangCat::]AppendCatalog ([, APP_NAME=*applicationName*] [, FILENAME=*filename*] [, APP_PATH=*applicationPath*])

IDLffLangCat::Cleanup - Performs all cleanup operations on the language catalog object.

OBJ_DESTROY, *Obj*

or

Obj->[IDLffLangCat::]Cleanup

IDLffLangCat::GetProperty - Retrieves a property or group of properties for an IDLffLangCat object.

Obj->[IDLffLangCat::]GetProperty [, PROPERTY=*variable*]

IDLffLangCat::Init - Initializes an IDLffLangCat object.

Result = OBJ_NEW('IDLffLangCat', *Language* [, *PROPERTY*=*value*] [, /CONTINUE_ON_ERROR]

IDLffLangCat::Query - Returns the string or string array that corresponds to all supplied key values.

Result = *Obj*->[IDLffLangCat::]Query(*Key* [, DEFAULT_STRING=*string*])

IDLffLangCat::SetProperty - Sets the value of a property or group of properties for an IDLffLangCat object.

Obj->[IDLffLangCat::]SetProperty [, PROPERTY=*variable*]

IDLffMrSID - Object class used to query information about and load image data from a MrSID (.sid) image file. No superclasses. No subclasses.

Properties: [, CHANNELS{Get}=*nChannels*] [, DIMENSIONS{Gets}=*Dims*] [, GEO_VALID{Get}=*geoValid*] [, GEO_PROJTYPE{Get}=*geoProjType*] [, GEO_ORIGIN{Get}=*geoOrigin*] [, GEO_RESOLUTION{Get}=*geoRes*] [, LEVELS{Get}=*Levels*] [, PIXEL_TYPE{Get}=*pixelType*] [, /QUIET{Init}] [, TYPE{Get}=*strType*]

IDLffMrSID::Cleanup - Deletes all MrSID objects, closing the MrSID file in the process.

OBJ_DESTROY, *Obj*

or

Obj->[IDLffMrSID::]Cleanup

IDLffMrSID::GetDimsAtLevel - Retrieves the dimensions of the image at a given level.

Dims = *Obj*->[IDLMrSID::]GetDimsAtLevel (*Level*)

IDLffMrSID::GetImageData - Returns the image data from the MrSID file.

ImageData = *Obj*->[IDLMrSID::]GetImageData ([, LEVEL = *lvl*] [, SUB_RECT = *rect*])

IDLffMrSID::GetProperty - Query properties associated with the MrSID image.

Obj->[IDLMrSID::]GetProperty[, *PROPERTY*=*variable*]

IDLffMrSID::Init - Initializes an IDLffMrSID object containing the image data from a MrSID image file.

Result = OBJ_NEW('IDLffMrSID', *Filename* [, *PROPERTY*=*value*])

IDLffShape - Contains geometry, connectivity and attributes for graphics primitives accessed from ESRI Shapefiles. No superclass. No subclasses.

Properties: [, /DBF_ONLY{Init}] [, ENTITY_TYPE{Init}=*'Value'*] [, /UPDATE{Init}]

IDLffShape::AddAttribute - Adds an attribute to a shapefile.

Obj->[IDLffShape::]AddAttribute, *Name*, *Type*, *Width* [, PRECISION=*integer*]

IDLffShape::Cleanup - Performs all cleanup on a Shapefile object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLffShape::]Cleanup

IDLffShape::Close - Closes a Shapefile.

Obj->[IDLffShape::]Close

IDLffShape::DestroyEntity - Frees memory associated with the entity structure.

Obj->[IDLffShape::]DestroyEntity, *Entity*

IDLffShape::GetAttributes - Retrieves the attributes for the entities you specify from a Shapefile.

Result = *Obj*->[IDLffShape::]GetAttributes([*Index*] [, /ALL] [, /ATTRIBUTE_STRUCTURE])

IDLffShape::GetEntity - Returns an array of entity structures from a Shapefile.

Result = *Obj*->[IDLffShape::]GetEntity([*Index*] [, /ALL] [, /ATTRIBUTES])

IDLffShape::GetProperty - Returns the values of properties associated with a Shapefile object.

Obj->[IDLffShape::]GetProperty [, ATTRIBUTE_INFO=*variable*] [, ATTRIBUTE_NAMES=*variable*] [, ENTITY_TYPE=*variable*] [, FILENAME=*variable*] [, IS_OPEN=*variable*] [, N_ATTRIBUTES=*variable*] [, N_ENTITIES=*variable*] [, N_RECORDS=*variable*]

IDLffShape::Init - Initializes or constructs a Shapefile object.

Result = OBJ_NEW('IDLffShape' [, *Filename*] [, *PROPERTY*=*value*])

IDLffShape::Open - Opens a specified Shapefile.

Result = *Obj*->[IDLffShape::]Open(*'Filename'* [, /DBF_ONLY] [, /UPDATE] [, ENTITY_TYPE=*'value'*])

IDLffShape::PutEntity - Inserts an entity into the Shapefile object.

Obj->[IDLffShape::]PutEntity, *Data*

IDLffShape::SetAttributes - Modifies the attributes for a specified entity in a Shapefile object.

Obj->[IDLffShape::]SetAttributes, *Index*, *Attribute_Num*, *Value*

or

Obj->[IDLffShape::]SetAttributes, *Index*, *Attributes*

IDLffXMLDOM Classes - Represents classes that provide support for IDL's XML Document Object Model (DOM).

IDLffXMLDOMAttr - Represents an attribute that is a part of an element object in an XML document. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMAttr:Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMAttr::GetName - Returns the attribute node's name.

Result = *Obj*->[IDLffXMLDOMAttr::]GetName()

IDLffXMLDOMAttr::GetSpecified - Returns a scalar integer indicating how the attribute node's value was set.

Result = *Obj*->[IDLffXMLDOMAttr::]GetSpecified()

IDLffXMLDOMAttr::GetValue - Returns the attribute node's value.

Result = *Obj*->[IDLffXMLDOMAttr::]GetValue()

IDLffXMLDOMAttr::SetValue - Sets the attribute node's value.

Obj->[IDLffXMLDOMAttr::]SetValue, *Value*

IDLffXMLDOMCDATASection - Used to escape blocks of text in an XML document containing text that would otherwise be regarded as market. Subclass of IDLffXMLDOMNode., IDLffXMLDOMCharacterData, and IDLffXMLDOMText.

IDLffXMLDOMCDATASection::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMCharacterData - Extends the IDLffXMLDOM class with methods for accessing character data in the DOM tree. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMCharacterData::AppendData - Appends a string to the node's character data.

Obj->[IDLffXMLDOMCharacterData::]AppendData, *String*

IDLffXMLDOMCharacterData::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMCharacterData::DeleteData - Deletes a number of characters from the node's character data, starting at an offset.

Obj->[IDLffXMLDOMCharacterData::]DeleteData, *Offset*, *Count*

IDLffXMLDOMCharacterData::GetData - Returns the node's character data.

Result = *Obj*->[IDLffXMLDOMCharacterData::]GetData()

IDLffXMLDOMCharacterData::GetLength - Returns the number of characters in the node.

Result = *Obj*->[IDLffXMLDOMCharacterData::]GetLength()

IDLffXMLDOMCharacterData::InsertData - Inserts a string in the node's character data, starting at an offset.

Obj->[IDLffXMLDOMCharacterData::]InsertData, *Offset*, *String*

IDLffXMLDOMCharacterData::ReplaceData - Replaces a number of characters, starting at an offset in the node's character data, with a string.

Obj->[IDLffXMLDOMCharacterData::]ReplaceData, *Offset*, *Count*, *String*

IDLffXMLDOMCharacterData::SetData - Sets the node's character data to a string

Obj->[IDLffXMLDOMCharacterData::]SetData, *String*

IDLffXMLDOMCharacterData::SubstringData - Returns a string composed of a substring of the node's character data.

Result = *Obj*->[IDLffXMLDOMCharacterData::]SubstringData (*Offset*, *Count)*

IDLffXMLDOMComment - Represents the content of a comment (characters between "<!--" and "-->!") in an XML document. Subclass of IDLffXMLDOMCharacterData and IDLffXMLDOMText.

IDLffXMLDOMComment::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMDocument - Represents the entire XML document as the document tree's root and by providing primary access to the document's data. Subclass of IDLffXMLDOMNode.

Properties: [, NODE_DESTRUCTION_POLICY{Get, Init, Set}=0 | 1]

Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree.

OBJ_DESTROY, *Obj*

or

Obj->[IDLffXMLDOMDocument::]Cleanup

IDLffXMLDOMDocument::CreateAttribute - Creates and names an attribute node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateAttribute(*Name*)

IDLffXMLDOMDocument::CreateCDATASection - Creates and fills a CDATASection node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateCDATASection(*String*)

IDLffXMLDOMDocument::CreateComment - Creates and fills a comment node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateComment, *String*)

IDLffXMLDOMDocument::CreateDocumentFragment - Creates a document fragment node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateDocumentFragment()

IDLffXMLDOMDocument::CreateElement - Creates and names with a tag name an element node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateElement(*TagName*)

IDLffXMLDOMDocument::CreateEntityReference - Creates and names an entity reference node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateEntityReference(*Name*)

IDLffXMLDOMDocument::CreateNodeIterator - Creates and names an instance of an IDLffXMLDOMNoteIterator object.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateNodeIterator(*Name* [, FILTER_NAME=*string*]
[, FILTER_USERDATA=*variable*]
[, WHAT_TO_SHOW=*value*])

IDLffXMLDOMDocument::CreateProcessingInstruction- Creates and stores two strings in a ProcessingInstruction node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateProcessingInstruction(*Target*, *Data*)

IDLffXMLDOMDocument::CreateTextNode - Creates and fills a text node owned by the XML document.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateTextNode(*String*)

IDLffXMLDOMDocument::CreateTreeWalker - Creates an instance of an IDLffXMLDOMTreeWalker object.

Result = *Obj*->[IDLffXMLDOMDocument::]
CreateTreeWalker(*RootNode* [, FILTER_NAME=*string*]
[, FILTER_USERDATA=*variable*]
[, WHAT_TO_SHOW=*value*])

IDLffXMLDOMDocument::GetDoctype - Creates an instance of IDLffXMLDOMDocumentType.

Result = *Obj*->[IDLffXMLDOMDocument::]
GetDoctype()

IDLffXMLDOMDocument::GetDocumentElement - Creates an instance of IDLffXMLDOMElement.

Result = *Obj*->[IDLffXMLDOMDocument::]
GetDocumentElement()

IDLffXMLDOMDocument::GetElementsByTagName - Creates an IDLffXMLDOMNodeList object containing all element nodes in the XML document with the specified tag name.

Result = *Obj*->[IDLffXMLDOMDocument::]
GetElementsByTagName(*Tagname*)

IDLffXMLDOMDocument::Init - Initializes the object.

Obj = OBJ_NEW('IDLffXMLDOMDocument'
[, /EXCLUDE_IGNORABLE_WHITESPACE]
[,/EXPAND_ENTITY_REFERENCES]
[, FILENAME=*string*] [, MSG_ERROR=*string*]
[, MSG_FATAL=*string*] [, MSG_WARNING=*string*]
[, /QUIET] [, SCHEMA_CHECKING=*value*]
[, VALIDATION_MODE=*value*])

or

Result = *Obj*->[IDLffXMLDOMDocument::]Init(
[, /EXCLUDE_IGNORABLE_WHITESPACE]
[, /EXPAND_ENTITY_REFERENCES]
[, FILENAME=*string*] [, MSG_ERROR=*string*]
[, MSG_FATAL=*string*] [, MSG_WARNING=*string*]
[, /QUIET] [, SCHEMA_CHECKING=*value*]
[, VALIDATION_MODE=*value*])

(Only in a subclass's Init method)

IDLffXMLDOMDocument::Load - Loads and parses XML data from a specified source; creates a DOM document tree (accessed through this object) from the data.

Obj->[IDLffXMLDOMDocument::]Load
[, /EXCLUDE_IGNORABLE_WHITESPACE]
[, /EXPAND_ENTITY_REFERENCES]
[, FILENAME=*string*] [, MSG_ERROR=*string*]
[, MSG_FATAL=*string*] [, MSG_WARNING=*string*]
[, /QUIET] [, SCHEMA_CHECKING=*value*]
[, VALIDATION_MODE=*value*]

IDLffXMLDOMDocument::Save - Serializes the current DOM document and writes it to an output source.

Obj->[IDLffXMLDOMDocument::]Save
[, ENCODING=*string*]
[, /EXPAND_ENTITY_REFERENCES]
[, FILENAME=string] [, /PRETTY_PRINT]

IDLffXMLDOMDocumentFragment - References a document fragment node. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMDocumentFragment::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMDocumentType - References a DocumentType node. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMDocumentType:Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMDocumentType::GetEntities - Returns the external and internal entities declared in the DTD.

Result = *Obj*->[IDLffXMLDOMDocumentType::]
GetEntities()

IDLffXMLDOMDocumentType::GetName - Returns the DTD's name.

Result = *Obj*->[IDLffXMLDOMDocumentType::]
GetName()

IDLffXMLDOMDocumentType::GetNotations - Returns the notations declared in the DTD.

Result = *Obj*->[IDLffXMLDOMDocumentType::]
GetNotations()

IDLffXMLDOMElement - References an element node. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMElement::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMElement::GetAttribute - Returns the value of the named attribute.

Result = *Obj*->[IDLffXMLDOMElement::]GetAttribute
(*Name*)

IDLffXMLDOMElement::GetAttributeNode - Creates an named IDLffXMLDOMAttr object.

Result = *Obj*->[IDLffXMLDOMElement::]
GetAttributeNode(*Name*)

IDLffXMLDOMElement::GetElementsByTagName - Creates an IDLffXMLDOMNodeList object containing all element nodes in the XML document with the specified tag name.

Result = *Obj*->[IDLffXMLDOMElement::]
GetElementsByTagName(*TagName*)

IDLffXMLDOMElement::GetTagName - Returns the element's name.

Result = *Obj*->[IDLffXMLDOMElement::]GetTagName()

IDLffXMLDOMElement::RemoveAttribute - Removes the named attribute from the element node.

Result = *Obj*->[IDLffXMLDOMElement::]
RemoveAttribute(*Name*)

IDLffXMLDOMElement::RemoveAttributeNode -
Removes the named attribute node from the element node.

Result = *Obj*->[IDLffXMLDOMElement::]
RemoveAttributeNode(*OldAttr*)

IDLffXMLDOMElement::SetAttribute - Adds and sets a new attribute to the element node.

Result = *Obj*->[IDLffXMLDOMElement::]
SetAttribute(*Name, Value)*

IDLffXMLDOMElement::SetAttributeNode - Adds a new attribute to the element node.

Result = *Obj*->[IDLffXMLDOMElement::]
SetAttributeNode(*NewAttr*)

IDLffXMLDOMEntity - References a parsed or unparsed entity in an XML document. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMEntity::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMEntity::GetNotationName - Returns the notation's name for the entity.

Result = *Obj*->[IDLffXMLDOMEntity::]
GetNotationName()

IDLffXMLDOMEntity::GetPublicId - Returns the entity's public ID if specified.

Result = *Obj*->[IDLffXMLDOMEntity::]GetPublicId()

IDLffXMLDOMEntity::GetSystemId - Returns the entity's system ID if specified.

Result = *Obj*->[IDLffXMLDOMEntity::]GetSystemId()

IDLffXMLDOMEntityReference - References an entity reference node in an XML document. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMEntityReference::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMNamedNodeMap - Container for nodes that uses node names for access. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMNamedNodeMap::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMNamedNodeMap::GetLength - Returns the number of nodes in the named node map.

Result = *Obj*->[IDLffXMLDOMNamedNodeMap::]
GetLength()

IDLffXMLDOMNamedNodeMap::GetNamedItem - Returns an object reference to the node that contains the named item.

Result = *Obj*->[IDLffXMLDOMNamedNodeMap::]GetNamedItem(*Name*)

IDLffXMLDOMNamedNodeMap::Item - Returns an object reference to the node that contains the indexed item.

Result = *Obj*->[IDLffXMLDOMNamedNodeMap::]Item(*Index)*

IDLffXMLDOMNamedNodeMap::RemoveNamedItem - Removes the specified node from the named node map.

Result = *Obj*->[IDLffXMLDOMNamedNodeMap::]RemoveNamedItem(*Name)*

IDLffXMLDOMNamedNodeMap::SetNamedItem - Adds and names a node to the named node map, replacing an existing node with the specified name if necessary.

Result = *Obj*->[IDLffXMLDOMNamedNodeMap::]SetNamedItem(*Name*)

IDLffXMLDOMNode - Abstract superclass for other IDLffXMLDOM node classes. No superclasses.

IDLffXMLDOMNode::AppendChild - Adds a new node at the end of the calling node's children list.

Result = *Obj*->[IDLffXMLDOMNode::]AppendChild(*NewChild*)

IDLffXMLDOMNode::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMNode::CloneNode - Creates a copy of the calling node; the cloned node has no parent.

Result = *Obj*->[IDLffXMLDOMNode::]CloneNode([, DEEP=*value*])

IDLffXMLDOMNode::GetAttributes - Creates a named node map used to access an element object's attributes.

Result = *Obj*->[IDLffXMLDOMNode::]GetAttributes()

IDLffXMLDOMNode::GetChildNodes - Creates a node list used to access the calling node's children.

Result = *Obj*->[IDLffXMLDOMNode::]GetChildNodes()

IDLffXMLDOMNode::GetFirstChild - Creates an IDLffXMLDOM node that refers to the first child in the DOM tree.

Result = *Obj*->[IDLffXMLDOMNode::]GetFirstChild()

IDLffXMLDOMNode::GetLastChild - Creates an IDLffXMLDOM node that refers to the last child in the DOM tree.

Result = *Obj*->[IDLffXMLDOMNode::]GetLastChild()

IDLffXMLDOMNode::GetNextSibling - Creates an IDLffXMLDOM node that refers to the next sibling in the DOM tree.

Result = *Obj*->[IDLffXMLDOMNode::]GetNextSibling()

IDLffXMLDOMNode::GetNodeName - Returns the node's name, depending on the subclass type.

Result = *Obj*->[IDLffXMLDOMNode::]GetNodeName()

IDLffXMLDOMNode::GetNodeType - Returns the node's type.

Result = *Obj*->[IDLffXMLDOMNode::]GetNodeType()

IDLffXMLDOMNode::GetNodeValue - Returns the node's value, depending on the subclass type.

Result = *Obj*->[IDLffXMLDOMNode::]GetNodeValue()

IDLffXMLDOMNode::GetOwnerDocument - Returns an object reference to the IDLffXMLDOMDocument used to create the node.

Result = *Obj*->[IDLffXMLDOMNode::]GetOwnerDocument()

IDLffXMLDOMNode::GetParentNode - Creates an IDLffXMLDOM node that refers to the parent in the DOM tree.

Result = *Obj*->[IDLffXMLDOMNode::]GetParentNode()

IDLffXMLDOMNode::GetPreviousSibling - Creates an IDLffXMLDOM node that refers to the previous sibling in the DOM tree.

Result = *Obj*->[IDLffXMLDOMNode::]GetPreviousSibling()

IDLffXMLDOMNode::HasChildNodes - Returns a value indicating whether the calling node has children.

Result = *Obj*->[IDLffXMLDOMNode::]HasChildNodes()

IDLffXMLDOMNode::InsertBefore - Inserts a new node into the calling node's children before a reference node.

Result = *Obj*->[IDLffXMLDOMNode::]InsertBefore(*NewChild* [, *RefChild*])

IDLffXMLDOMNode::RemoveChild - Removes an existing node from the calling node's child list.

Result = *Obj*->[IDLffXMLDOMNode::]RemoveChild (*OldChild*)

IDLffXMLDOMNode::ReplaceChild - Replaces an existing child node with a new node in the calling node's child list.

Result = *Obj*->[IDLffXMLDOMNode::]ReplaceChild(*NewChild*, *OldChild*)

IDLffXMLDOMNode::SetNodeValue - Sets the node's value to the contents of a string.

Result = *Obj*->[IDLffXMLDOMNode::]SetNodeValue(*NodeValue*)

IDLffXMLDOMNodeIterator - Allows iterative navigation of an IDLffXMLDOM tree. No superclasses.

IDLffXMLDOMNodeIterator::Cleanup - Destroys both the node-iterator object and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMNodeIterator::NextNode - Returns an object reference to the DOM tree node that appears after the iterator's current position in document-order traversal.

Result = *Obj*->[IDLffXMLDOMNodeIterator::]NextNode()

IDLffXMLDOMNodeIterator::PreviousNode - Returns an object reference to the DOM tree node that appears before the iterator's current position in document-order traversal.

Result = *Obj*->[IDLffXMLDOMNodeIterator::]PreviousNode()

IDLffXMLDOMNodeList - Container for nodes. No superclasses.

IDLffXMLDOMNodeList::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMNodeList::GetLength - Returns the number of nodes in the node list.

Result = *Obj*->[IDLffXMLDOMNodeList::]GetLength()

IDLffXMLDOMNodeList::Item - Returns an object reference to the node that contains the indexed item.

Result = *Obj*->[IDLffXMLDOMNodeList::]Item(*Index*)

IDLffXMLDOMNotation - References a notation in the DTD. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMNotation::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMNotation::GetPublicID - Returns the notation's public ID.

Result = *Obj*->[IDLffXMLDOMNotation::]GetPublicID()

IDLffXMLDOMNotation::GetSystemID - Returns the notation's system ID.

Result = *Obj*->[IDLffXMLDOMNotation::]GetSystemID()

IDLffXMLDOMProcessingInstruction - References a processing instruction node in the XML document. Subclass of IDLffXMLDOMNode.

IDLffXMLDOMProcessingInstruction::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMProcessingInstruction::GetData - Returns the content of the processing instruction.

Result = *Obj*->[IDLffXMLDOMProcessingInstruction::]GetData()

IDLffXMLDOMProcessingInstruction::GetTarget - Returns the target of the processing instruction.

Result = *Obj*->[IDLffXMLDOMProcessingInstruction::]GetTarget()

IDLffXMLDOMProcessingInstruction::SetData - Sets the content of the processing instruction.

Obj->[IDLffXMLDOMProcessingInstruction::]SetData, *Content*

IDLffXMLDOMText - References a text node in the XML document. Subclass of IDLffXMLDOMNode and IDLffXMLDOMCharacterData.

IDLffXMLDOMText::Cleanup - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMText::IsIgnorableWhitespace - Indicates whether the text node contains ignorable whitespace.

Result = *Obj*->[IDLffXMLDOMText::]IsIgnorableWhitespace()

IDLffXMLDOMText::SplitText - Breaks a text node into two sibling nodes in the same tree, splitting the text at the specified offset.

Result = *Obj*->[IDLffXMLDOMText::]SplitText(*Offset*)

IDLffXMLDOMTreeWalker - Allows tree-walking navigation of an IDLffXMLDOM tree. No superclasses.

IDLffXMLDOMTreeWalker::Cleanup - Destroys both the node-iterator object and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ_DESTROY, *Obj*

IDLffXMLDOMTreeWalker::FirstChild - Returns an object reference to the DOM tree node that is the first child of the node to which the walker is currently pointing.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]FirstChild()

IDLffXMLDOMTreeWalker::GetCurrentNode - Returns an object reference to the DOM tree node to which the walker is currently pointing.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]GetCurrentNode()

IDLffXMLDOMTreeWalker::LastChild - Returns an object reference to the DOM tree node that is the last child of the node to which the walker is currently pointing.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]LastChild()

IDLffXMLDOMTreeWalker::NextNode - Returns an object reference to the DOM tree node visited next after the walker's current node in document-order traversal.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]NextNode()

IDLffXMLDOMTreeWalker::NextSibling - Returns an object reference to the DOM tree node that is the next sibling of the node to which the walker is currently pointing.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]NextSibling()

IDLffXMLDOMTreeWalker::ParentNode - Returns an object reference to the DOM tree node that is the parent of the node to which the walker is currently pointing.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]ParentNode()

IDLffXMLDOMTreeWalker::PreviousNode - Returns an object reference to the DOM tree node visited before the walker's current node in document-order traversal.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]PreviousNode()

IDLffXMLDOMTreeWalker::PreviousSibling - Returns an object reference to the DOM tree node that is the previous sibling of the node to which the walker is currently pointing.

Result = *Obj*->[IDLffXMLDOMTreeWalker::]PreviousSibling()

IDLffXMLDOMTreeWalker::SetCurrentNode - Sets the walker's current node to the specified node.

Obj->[IDLffXMLDOMTreeWalker::]SetCurrentNode(*Current Node*)

IDLffXMLSAX - Represents an XML SAX Level 2 parser. No superclasses. In order to use this class, you *must* write your own subclass.

Properties: [, FILENAME{Get}=*variable*]
[, /NAMESPACE_PREFIXES{Get, Init, Set}]
[, PARSER_LOCATION{Get}=*variable*]
[, PARSER_PUBLICID{Get}=*variable*]
[, PARSER_URI{Get}=*variable*]
[, SCHEMA_CHECKING{Get, Init, Set}=[0,1,2]]
[, VALIDATION_MODE{Get, Init, Set}=[0,1,2]]

IDLffXMLSAX::AttributeDecl - Called when the parser detects an <!ATTLIST ...> declaration in a DTD.

Obj->[IDLffXMLSAX::]AttributeDecl, *eName*, *aName*, *Type*, *Mode*, *Value*

IDLffXMLSAX::Characters - Called when the parser detects text in the parsed document.

Obj->[IDLffXMLSAX::]Characters, *Chars*

IDLffXMLSAX::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj*

or

Obj->[IDLffXMLSAX::]Cleanup

IDLffXMLSAX::Comment - Called when the parser detects a comment section of the form <!-- ... -->.

Obj->[IDLffXMLSAX::]Comment, *Comment*

IDLffXMLSAX::ElementDecl - Called when the parser detects an <!ELEMENT ...> declaration in the DTD.

Obj->[IDLffXMLSAX::]ElementDecl, *Name*, *Model*

IDLffXMLSAX::EndCDATA - Called when the parser detects the end of a <![CDATA[...]]> text section.

Obj->[IDLffXMLSAX::]EndCDATA

IDLffXMLSAX::EndDocument - Called when the parser detects the end of the XML document.

Obj->[IDLffXMLSAX::]EndDocument

IDLffXMLSAX::EndDTD - Called when the parser detects the end of a Document Type Definition (DTD).

Obj->[IDLffXMLSAX::]EndDTD

IDLffXMLSAX::EndElement - Called when the parser detects the end of an element.

Obj->[IDLffXMLSAX::]EndElement, *URI*, *Local*, *qName*

IDLffXMLSAX::EndEntity - Called when the parser detects the end of an internal or external entity expansion.

Obj->[IDLffXMLSAX::]EndEntity, *Name*

IDLffXMLSAX::EndPrefixMapping - Called when a previously declared prefix mapping goes out of scope.

Obj->[IDLffXMLSAX::]EndPrefixMapping, *Prefix*

IDLffXMLSAX::Error procedure - Called when the parser detects and error that is not expected to be fatal.

Obj->[IDLffXMLSAX::]Error, *SystemID*, *LineNumber*, *ColumnNumber*, *Message*

IDLffXMLSAX::ExternalEntityDecl - Called when the parser detects an <!ENTITY ...> declarations in the DTD for a parsed external entity.

Obj->[IDLffXMLSAX::]ExternalEntityDecl, *Name*, *PublicID*, *SystemID*

IDLffXMLSAX::FatalError - Called when the parser detects a fatal error.

Obj->[IDLffXMLSAX::]FatalError, *SystemID*, *LineNumber*, *ColumnNumber*, *Message*

IDLffXMLSAX::GetProperty - Used to get the values of various properties of the parser.

Obj->[IDLffXMLSAX::]GetProperty
[, *PROPERTY*=*variable*]

IDLffXMLSAX::IgnorableWhitespace - Called when the parser detects whitespace that separates elements in an element content model.

Obj->[IDLffXMLSAX::]IgnorableWhitespace, *Chars*

IDLffXMLSAX::Init - Initializes an XML parser object.

Obj = OBJ_NEW('IDLffXMLSAX'
[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLffXMLSAX::]Init([*PROPERTY*=*value*])

IDLffXMLSAX::InternalEntityDecl - Called when the parser detects an <!ENTITY ...> declaration in a DTD for (parsed) internal entities.

Obj->[IDLffXMLSAX::]InternalEntitydecl, *Name*, *Value*

IDLffXMLSAX::NotationDecl - Called when the parser detects a <!NOTATION ...> declaration in a DTD.

Obj->[IDLffXMLSAX::]NotationDecl, *Name*, *PublicID*, *SystemID*

IDLffXMLSAX::ParseFile - Parses the specified XML file.

Obj->[IDLffXMLSAX::]ParseFile, *Filename*

IDLffXMLSAX::ProcessingInstruction - Called when the parser detects a processing instruction.

Obj->[IDLffXMLSAX::]ProcessingInstruction, *Target*, *Data*

IDLffXMLSAX::SetProperty - Used to set the values of various properties of the parser.

Obj->[IDLffXMLSAX::]SetProperty
[, *PROPERTY*=*value*]

IDLffXMLSAX::SkippedEntity - Called when the parser skips an entity *and* validation is not being performed.

Obj->[IDLffXMLSAX::]SkippedEntity, *Name*

IDLffXMLSAX::StartCDATA - Called when the parser detects the beginning of a <![CDATA[...]]> text section.

Obj->[IDLffXMLSAX::]StartCDATA

IDLffXMLSAX::StartDocument - Called when the parser begins processing a document, and before any data is processed.

Obj->[IDLffXMLSAX::]StartDocument

IDLffXMLSAX::StartDTD - Called when the parser detects the beginning of a Document Type Definition (DTD).

Obj->[IDLffXMLSAX::]StartDTD, *Name*, *PublicID*, *SystemID*

IDLffXMLSAX::StartElement - Called when the parser detects the beginning of an element.

Obj->[IDLffXMLSAX::]StartElement, *URI*, *Local*, *qName* [, *attName*, *attValue*]

IDLffXMLSAX::StartEntity - Called when the parser detects the start of an internal or external entity expansion.

Obj->[IDLffXMLSAX::]StartEntity, *Name*

IDLffXMLSAX::StartPrefixmapping - Called when the parser detects the beginning of a namespace declaration.

Obj->[IDLffXMLSAX::]StartPrefixmapping, *Prefix*, *URI*

IDLffXMLSAX::StopParsing - Used during a parse operation to halt the operation and cause the ParseFile method to return.

Obj->[IDLffXMLSAX::]StopParsing

IDLffXMLSAX::UnparsedEntityDecl - Called when the parser detects an <!ENTITY ...> declaration that includes the NDATA keyword, indicating that the entity is not meant to be parsed.

Obj->[IDLffXMLSAX::]UnparsedEntityDec, *Name*, *PublicID*, *SystemID*, *Notation*

IDLffXMLSAX::Warning - Called when the parser detects a problem during processing.

Obj->[IDLffXMLSAX::]Warning, *SystemID*, *LineNumber*, *ColumnNumber*, *Message*

IDLgrAxis - Represents a single vector that may include a set of tick marks, tick labels, and a title.

Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, AM_PM{Get, Init, Set}=*array*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB_vector*] [, CRANGE{Get}=*variable*] [, DAYS_OF_WEEK{Get, Init, Set}=*array*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DIRECTION{Get, Init, Set}=*integer*] [, /EXACT{Get, Init, Set}] [, /EXTEND{Get, Init, Set}] [, GRIDSTYLE{Get, Init, Set}=*integer*{0 to 6}or *[repeat*{1 to 255}, *bitmask]*] [, /HIDE{Get, Init, Set}] [, LOCATION{Get, Init, Set}=*[x, y]* or *[x, y, z]*] [, /LOG{Get, Init, Set}] [, MAJOR{Get, Init, Set}=*integer*] [, MINOR{Get, Init, Set}=*integer*] [, MONTHS{Get, Init, Set}=*array*] [, /NOTEXT{Get, Init, Set}] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT{Get}=*variable*] [, RANGE{Get, Init, Set}=*[min, max]*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, SUBTICKLEN{Get, Init, Set}=*value*] [, TEXTALIGNMENTS{Get, Init, Set}=*[horiz*{0.0 to 1.0}, *vert*{0.0 to 1.0}*]*] [, TEXTBASELINE{Get, Init, Set}=*vector*] [, TEXTPOS{Get, Init, Set}={0 | 1}] [, TEXTUPDIR{Get, Init, Set}=*vector*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, TICKDIR{Get, Init, Set}={0 | 1}] [, TICKFORMAT{Get, Init, Set}=*string or array of strings*]

IDLgrAxis Properties - *continued*

[, TICKFRMTDATA{Get, Init, Set}=*value*] [, TICKINTERVAL{Get, Init, Set}=*value*] [, TICKLAYOUT{Get, Init, Set}=*scalar*] [, TICKLEN{Get, Init, Set}=*value*] [, TICKTEXT{Get, Init, Set}=*objref or vector*] [, TICKUNITS{Get, Init, Set}=*string or a vector of strings*] [, TICKVALUES{Get, Init, Set}=*vector*] [, TITLE{Get, Init, Set}=*objref*] [, /USE_TEXT_COLOR{Get, Init, Set}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrAxis::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj->*[IDLgrAxis::]Cleanup

IDLgrAxis::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj->*[IDLgrAxis::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref*])

IDLgrAxis::GetProperty - Retrieves the value of a property or group of properties for the axis.

Obj->[IDLgrAxis::]GetProperty [, *PROPERTY*=*variable*]

IDLgrAxis::Init - Initializes an axis object.

Obj = OBJ_NEW('IDLgrAxis' [, *Direction*] [, *PROPERTY*=*value*]) or

Result = *Obj->*[IDLgrAxis::]Init([*Direction*] [, *PROPERTY*=*value*])

IDLgrAxis::SetProperty - Sets the value of a property or group of properties for the axis.

Obj->[IDLgrAxis::]SetProperty [, *PROPERTY*=*value*]

IDLgrBuffer - An in-memory, off-screen destination object.

Properties: [, ALL{Get}=*variable*] [, COLOR_MODEL{Get, Init}={0 | 1}] [, DIMENSIONS{Get, Init, Set}=*[width, height]*] [, GRAPHICS_TREE{Get, Init, Set}=*objref*] [, IMAGE_DATA{Get}=*variable*] [, N_COLORS{Get, Init}=*integer*{2 to 256}] [, PALETTE{Get, Init, Set}=*objref*] [, QUALITY{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, RESOLUTION{Get, Init, Set}=*[xres, yres]*] [, SCREEN_DIMENSIONS{Get}=*variable*] [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}] [, ZBUFFER_DATA{Get}=*variable*]

IDLgrBuffer::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj->*[IDLgrBuffer::]Cleanup

IDLgrBuffer::Draw - Draws picture to this graphics destination.

Obj->[IDLgrBuffer::]Draw [, *Picture*] [, CREATE_INSTANCE={1 | 2}] [, DRAW_INSTANCE={1 | 2}]

IDLgrBuffer::Erase - Erases this graphics destination.

Obj->[IDLgrBuffer::]Erase [, COLOR=*index or RGB vector*]

IDLgrBuffer::GetContiguousPixels - Returns an array of long integers whose length is equal to the number of colors available in the index color mode (value of the N_COLORS property).

Return = *Obj*->[IDLgrBuffer::]GetContiguousPixels( )

IDLgrBuffer::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Obj–>[IDLgrBuffer::]GetDeviceInfo [, ALL=*variable*]
[, MAX_NUM_CLIP_PLANES=*variable*]
[, MAX_TEXTURE_DIMENSIONS=*variable*]
[MAX_TILE_DIMENSIONS=*variable*]
[, MAX_VIEWPORT_DIMENSIONS=*variable*]
[, NAME=*variable*] [, NUM_CPUS=*variable*]
[, VENDOR=*variable*] [, VERSION=*variable*]

IDLgrBuffer::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj*->[IDLgrBuffer::]GetFontnames(*FamilyName*
[, IDL_FONTS={0 | 1 | 2 }] [, STYLES=*string*])

IDLgrBuffer::GetProperty - Retrieves the value of a property or group of properties for the buffer.

Obj->[IDLgrBuffer::]GetProperty
[, *PROPERTY*=*variable*]

IDLgrBuffer::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the buffer.

Result = *Obj*->[IDLgrBuffer::]GetTextDimensions(
TextObj [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrBuffer::Init - Initializes the buffer object.

Obj = OBJ_NEW('IDLgrBuffer'[, *PROPERTY*=*value*])
or *Result* = *Obj*->[IDLgrBuffer::]Init(
[, *PROPERTY*=*value*])

IDLgrBuffer::PickData - Maps a point in the 2D device space of the buffer to a point in the 3D data space of an object tree.

Result = *Obj*->[IDLgrBuffer::]PickData(*View, Object,*
Location, XYZLocation
[, DIMENSIONS=*[width,height]*] [, PATH=*objref(s)*]
[, PICK_STATUS=*variable*])

IDLgrBuffer::QueryRequiredTiles - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

Result = *Obj*->[IDLgrWindow::]QueryRequiredTiles
(*View, Image* [, COUNT=*variable*]
[, ALL_VISIBLE=*value*])

IDLgrBuffer::Read - Reads an image from a buffer.

Result = *Obj*->[IDLgrBuffer::]Read( )

IDLgrBuffer::Select - Returns a list of objects selected at a specified location.

Result = *Obj*->[IDLgrBuffer::]Select(*Picture, XY*
[, DIMENSIONS=*[width, height]*] [, /ORDER]
[, SUB_SELECTION=*variable*]
[, UNITS={0 | 1 | 2 | 3}])

IDLgrBuffer::SetProperty - Sets the value of a property or group of properties for the buffer.

Obj->[IDLgrBuffer::]SetProperty[, *PROPERTY*=*value*]

IDLgrClipboard - A destination object representing the native clipboard.

Properties: [, ALL{Get}=*variable*]
[, COLOR_MODEL{Get, Init}={0 | 1}]
[, DIMENSIONS{Get, Init, Set}=*[width, height]*]
[, GRAPHICS_TREE{Get, Init, Set}=*objref*]
[, N_COLORS{Get, Init}=*integer*{2 to 256}]
[, PALETTE{Get, Init, Set}=*objref*]
[, QUALITY{Get, Init, Set}={0 | 1 | 2}]
[, /REGISTER_PROPERTIES{Get, Init, Set}]
[, RESOLUTION{Get, Init, Set}=*[xres, yres]*]
[, SCREEN_DIMENSIONS{Get}=*variable*]
[, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]

IDLgrClipboard::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj* or *Obj*->[IDLgrClipboard::]Cleanup

IDLgrClipboard::Draw - Draws a picture to a graphics destination.

Obj->[IDLgrClipboard::]Draw [, *Picture*] [, /CMYK]
[, FILENAME=*string*] [, POSTSCRIPT=*value*]
[, VECT_SHADING={ 0 | 1 }]
[, VECT_SORTING={ 0 | 1 }]
[, VECT_TEXT_RENDER_METHOD={ 0 | 1 }]
[, VECTOR={ 0 | 1 }]

IDLgrClipboard::GetContiguousPixels - Returns array of long integers whose length is equal to the number of colors available in the index color mode (value of the N_COLORS property).

Return = *Obj*->[IDLgrClipboard::]GetContiguousPixels()

IDLgrClipboard::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Obj–>[IDLgrClipboard::]GetDeviceInfo [, ALL=*variable*]
[, MAX_NUM_CLIP_PLANES=*variable*]
[, MAX_TEXTURE_DIMENSIONS=*variable*]
[MAX_TILE_DIMENSIONS=*variable*]
[, MAX_VIEWPORT_DIMENSIONS=*variable*]
[, NAME=*variable*] [, NUM_CPUS=*variable*]
[, VENDOR=*variable*] [, VERSION=*variable*]

IDLgrClipboard::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj*->[IDLgrClipboard::]GetFontnames(
FamilyName [, IDL_FONTS={0 | 1 | 2}]
[, STYLES=*string*])

IDLgrClipboard::GetProperty - Retrieves the value of a property or group of properties for the clipboard buffer.

Obj->[IDLgrClipboard::]GetProperty
[, *PROPERTY*=*variable*]

IDLgrClipboard::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the clipboard buffer.

Result = *Obj*->[IDLgrClipboard::]GetTextDimensions(
TextObj [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrClipboard::Init - Initializes the clipboard object.

Obj = OBJ_NEW('IDLgrClipboard'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrClipboard::]Init([*PROPERTY*=*value*])

IDLgrClipboard::QueryRequiredTiles - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

Result = *Obj*->[IDLgrWindow::]QueryRequiredTiles (*View*, *Image* [, COUNT=*variable*] [, ALL_VISIBLE=*value*])

IDLgrClipboard::SetProperty - Sets the value of a property or group of properties for the clipboard buffer.

Obj->[IDLgrClipboard::]SetProperty [, *PROPERTY*=*value*]

IDLgrColorbar - Consists of a color-ramp with an optional framing box and annotation axis.

Properties: [, ALL{Get}=*variable*] [, BLUE_VALUES{Get, Init, Set}=*vector*] [, COLOR{Get, Init, Set}=*index or RGB vector*] [, DIMENSIONS{Get, Init, Set}=*[dx, dy]*] [, GREEN_VALUES{Get, Init, Set}=*vector*] [, /HIDE{Get, Init, Set}] [, MAJOR{Get, Init,Set}=*integer*] [, MINOR{Get, Init, Set}=*integer*] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT{Get}=*variable*] [, RED_VALUES{Get, Init, Set}=*vector*] [, SHOW_AXIS{Get, Init, Set}={0 | 1 | 2}] [, /SHOW_OUTLINE{Get, Init, Set}] [, SUBTICKLEN{Get, Init, Set}=*minorTickLength /majorTickLength*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, /THREED{Get, Init}] [, TICKFORMAT{Get, Init, Set}=*string*] [, TICKFRMTDATA{Get, Init, Set}=*value*] [, TICKLEN{Get, Init, Set}=*value*] [, TICKTEXT{Get, Init, Set}=*objref(s)*] [, TICKVALUES{Get, Init, Set}=*vector*] [, TITLE{Get, Init, Set}=*objref*] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrColorbar::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj* or *Obj*->[IDLgrColorbar::]Cleanup

IDLgrColorbar::ComputeDimensions - Retrieves the dimensions of a colorbar object for the given destination object.

Result = *Obj*->[IDLgrColorbar::]ComputeDimensions(*DestinationObj* [, PATH=*objref(s)*])

IDLgrColorbar::GetProperty - Retrieves the value of a property or group of properties for the colorbar.

Obj->[IDLgrColorbar::]GetProperty [, *PROPERTY*=*variable*]

IDLgrColorbar::Init - Initializes the colorbar object.

Obj = OBJ_NEW('IDLgrColorbar'[, *aRed*, *aGreen*, *aBlue*] [, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrColorbar::]Init([*aRed*, *aGreen*, *aBlue*] [, *PROPERTY*=*value*])

IDLgrColorbar::SetProperty - Sets the value of a property or group of properties for the colorbar.

Obj->[IDLgrColorbar::]SetProperty [, *PROPERTY*=*value*]

IDLgrContour - Draws a contour plot from an array or unstructured point data. No superclasses. No subclasses.

Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, AM_PM{Get, Init, Set}=*vector of two strings*] [, ANISOTROPY{Get, Init, Set}=*[x, y, z]*] [, C_COLOR{Get, Init, Set}=*vector*] [, C_FILL_PATTERN{Get, Init, Set}=*array of IDLgrPattern objects*] [, C_LABEL_INTERVAL{Get, Init, Set} =*vector*] [, C_LABEL_NOGAPS{Get, Init, Set}=*vector*] [, C_LABEL_OBJECTS{Get, Init, Set}=*array of object references*] [, C_LABEL_SHOW{Get, Init, Set}=*vector of integers*] [, C_LINESTYLE{Get, Init, Set}=*array of linestyles*] [, C_THICK{Get, Init, Set}=*float array*{each element 1.0 to 10.0}] [, C_USE_LABEL_COLOR{Get, Init, Set}= *vector of values*] [, C_USE_LABEL_ORIENTATION {Get, Init, Set}=vector of values] [, C_VALUE{Get, Init, Set}=*scalar or vector*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB vector*] [, DATA_VALUES{Get, Init, Set}=*vector or 2D array*] [, DAYS_OF_WEEK{Get, Init, Set}=*vector of seven strings*] [, DEPTH_OFFSET{Get, Init, Set}=*value*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, /DOUBLE_DATA{Init}] [, /DOUBLE_GEOM{Init}] [, /DOWNHILL{Get, Init, Set}] [, /FILL{Get, Init, Set}] [, GEOM{Get}=*variable*] [, GEOMX{Init, Set}=*vector or 2D array*] [, GEOMY{Init, Set}=*vector or 2D array*] [, GEOMZ{Init, Set}=*scalar, vector, or 2D array*] [, /HIDE{Get, Init, Set}] [, LABEL_FONT{Get, Init, Set}=*objref*] [, LABEL_FORMAT{Get, Init, Set}=*string*] [, LABEL_FRMTDATA{Init}=*value*] [, LABEL_UNITS{Get, Init, Set}=*string*] [, MAX_VALUE{Get, Init, Set}=*value*] [, MIN_VALUE{Get, Init, Set}=*value*] [, MONTHS{Get, Init, Set}=*vector of 12 values*] [, N_LEVELS{Get, Init, Set}=*value*] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT{Get}=*variable*] [, /PLANAR{Get, Init, Set}]

IDLgrContour Properties - *continued*

[, POLYGONS{Get, Init, Set}=*array of polygon descriptions*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, SHADE_RANGE{Get, Init, Set}=*[min, max]*] [, SHADING{Get, Init, Set}={0 |1}] [, TICKINTERVAL{Get, Set}=*value*] [, TICKLEN{Get, Init, Set}=*value*] [, USE_TEXT_ALIGNMENTS{Get, Init, Set}=*value*] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrContour::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrContour::]Cleanup

IDLgrContour::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object

Result = *Obj*->[IDLgrContour::]GetCTM ([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref*])

IDLgrContour::GetLabelInfo - Retrieves information about the labels for a contour

Obj->[IDLgrContour::]GetLabelInfo, *Destination*, *LevelIndex* [, LABEL_OFFSETS=*variable*] [, LABEL_POLYS=*variable*] [, LABEL_OBJECTS=*variable*]

IDLgrContour::GetProperty - Retrieves the value of a property or group of properties for the contour.

Obj->[IDLgrContour::]GetProperty [, *PROPERTY*=*variable*]

IDLgrContour::Init - Initializes the contour object.

Obj = OBJ_NEW('IDLgrContour'[, *Values*] [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLgrContour::]Init([*Values*] [, *PROPERTY*=*value*])

IDLgrContour::SetProperty - Sets the value of a property or group of properties for the contour.

Obj->[IDLgrContour::]SetProperty[, *PROPERTY*=*value*]

IDLgrFont - Represents a typeface, style, weight, and point size that may be associated with text objects.

Properties: [, ALL{Get}=*variable*] [, SIZE{Get, Init, Set}=*points*] [, SUBSTITUTE{Get, Init, Set}={'Helvetica' | 'Courier' | 'Times' | 'Symbol' | 'Hershey'}] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, UVALUE{Get, Init, Set}=*value*]

IDLgrFont::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrFont::]Cleanup

IDLgrFont::GetProperty - Retrieves the value of a property or group of properties for the font.

Obj->[IDLgrFont:]GetProperty [, *PROPERTY*=*variable*]

IDLgrFont::Init - Initializes the font object.

Obj = OBJ_NEW('IDLgrFont'[, *Fontname*] [, *PROPERTY*=*value*])

or

Result = *Obj*->[IDLgrFont::]Init([*Fontname*] [, *PROPERTY*=*value*])

IDLgrFont::SetProperty - Sets the value of a property or group of properties for the font.

Obj->[IDLgrFont:]SetProperty[, *PROPERTY*=*value*]

IDLgrImage - Represents a mapping from a 2D array of data values to a 2D array of pixel colors.

Properties: [, ALL{Get}=*variable*] [, ALPHA_CHANNEL=*value*] [, BLEND_FUNCTION{Get, Init, Set}=*vector*] [, CHANNEL{Get, Init, Set}=*hexadecimal bitmask*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, DATA{Get, Init, Set}=*nxm*, *2xnxm*, *3xnxm*, or *4xnxm array of image data*] [, DEPTH_OFFSET=*value*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DIMENSIONS{Get, Init, Set}=*[width, height]*] [, /GREYSCALE{Get, Init, Set}] [, /HIDE{Get, Init, Set}] [, INTERLEAVE{Get, Init, Set}={0 | 1 | 2}] [, /INTERPOLATE{Get, Init, Set}] [LOCATION{Get, Init, Set}=*[x, y] or [x, y, z]*] [, /ORDER{Get, Init, Set}] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT {Get}=*variable*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [,RENDER_METHOD=*value*] [, /RESET_DATA{Init, Set}] [, SHARE_DATA{Init, Set}=*objref*] [, SUB_RECT{Get, Init, Set}=*[x, y, xdim, ydim]*] [, TILE_COLOR{Get, Init, Set}=*[red, green, blue]*] [, TILE_CURRENT_LEVEL{Get, Init, Set}=*value*] [, TILE_DIMENSIONS{Get, Init, Set}=*[width, height]*] [, TILE_LEVEL_MODE{Get, Init, Set}={0 | 1 }] [, TILE_NUM_LEVELS{Get, Init, Set}=*value*] [, TILE_SHOW_BOUNDARIES{Get, Init, Set}={0 | 1 }] [, TILED_IMAGE_DIMENSIONS{Get, Init, Set}=*[width, height]*] [, /TILING{Get, Init, Set}] [, TRANSFORM_MODE={0 | 1}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrImage::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrImage::]Cleanup

IDLgrImage::DeleteTileData - Removes tile data stored in the cache.

Obj->[IDLgrImage::]DeleteTileData, *TileInfo*, [, /ALL]

IDLgrImage::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object.

Result = *Obj*->[IDLgrImage::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrImage::GetProperty - Retrieves the value of the property or group of properties for the image.

Obj->[IDLgrImage::]GetProperty [, *PROPERTY*=*variable*]

IDLgrImage::Init - Initializes the image object.

Obj = OBJ_NEW('IDLgrImage'[, *ImageData*] [, *PROPERTY*=*value*] [, /NO_COPY]) or *Result* = *Obj*->[IDLgrImage::]Init([*ImageData*] [, *PROPERTY*=*value*] [, /NO_COPY])

IDLgrImage::SetProperty - Sets the value of the property or group of properties for the image.

Obj->[IDLgrImage::]SetProperty[, *PROPERTY*=*value*] [, /NO_COPY]

IDLgrImage::SetTileData - Stores tile data in the tile data cache.

Obj->[IDLgrImage::]SetTileData, *TileInfo*, *TileData*, [, NO_FREE=*value*]

IDLgrLegend - Provides a simple interface for displaying a legend.

Properties: [, ALL{Get}=*variable*] [, BORDER_GAP{Get, Init, Set}=*value*] [, COLUMNS{Get, Init, Set}=*integer*] [, FILL_COLOR{Get, Init, Set}=*index or RGB vector*] [, FONT{Get, Init, Set}=*objref*] [, GAP{Get, Init, Set}=*value*] [, GLYPH_WIDTH{Get, Init, Set}=*value*] [, /HIDE{Get, Init, Set}] [, ITEM_COLOR{Get, Init, Set}=*array of colors*] [, ITEM_LINESTYLE{Get, Init, Set}=*integer array*] [, ITEM_NAME{Get, Init, Set}=*string array*] [, ITEM_OBJECT{Get, Init, Set}=*array of IDLgrSymbol or IDLgrPattern objrefs*] [, ITEM_THICK{Get, Init, Set}=*float array*{each element 1.0 to 10.0}] [, ITEM_TYPE{Get, Init, Set}=*int array*{each element 0 or 1}] [, OUTLINE_COLOR{Get, Init, Set}=*index or RGB vector*] [, OUTLINE_THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, PARENT{Get}=*variable*] [, RECOMPUTE{Set}={0 | 1}{0 prevents recompute, 1 is the default}] [, /SHOW_FILL{Get, Init, Set}] [, /SHOW_OUTLINE{Get, Init, Set}] [, TEXT_COLOR{Get, Init, Set}=*index or RGB vector*] [, TITLE{Get, Init, Set}=*objref*] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrLegend::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrLegend::]Cleanup

IDLgrLegend::ComputeDimensions - Retrieves the dimensions of a legend object for the given destination object.

Result = *Obj*->[IDLgrLegend::]ComputeDimensions(*DestinationObj* [, PATH=*objref(s)*])

IDLgrLegend::GetProperty - Retrieves the value of a property or group of properties for the legend.

Obj->[IDLgrLegend::]GetProperty [, *PROPERTY*=*variable*]

IDLgrLegend::Init - Initializes the legend object.

Obj = OBJ_NEW('IDLgrLegend'[, *aItemNames*] [, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrLegend::]Init([*aItemNames*] [, *PROPERTY*=*value*])

IDLgrLegend::SetProperty - Sets the value of a property or group of properties for the legend.

Obj->[IDLgrLegend::]SetProperty[, *PROPERTY*=*value*]

IDLgrLight - Represents a source of illumination for 3D graphic objects.

Properties: [, ALL{Get}=*variable*] [, ATTENUATION{Get, Init, Set}=*[constant, linear, quadratic]*] [, COLOR{Get, Init, Set}=*[R, G, B]*] [, CONEANGLE{Get, Init, Set}=*degrees*] [, DIRECTION{Get, Init, Set}=*3-element vector*] [, FOCUS{Get, Init, Set}=*value*] [, /HIDE{Get, Init, Set}] [, INTENSITY{Get, Init, Set}=*value*{0.0 to 1.0}] [, LOCATION{Get, Init, Set}=*[x, y, z]*] [, PARENT{Get}=*variable*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, TYPE{Get, Init, Set}={0 | 1 | 2 | 3}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, ZCOORD_CONV{Get, Init, Set}=*vector*]

IDLgrLight::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrLight::]Cleanup

IDLgrLight::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object.

Result = *Obj*->[IDLgrLight::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrLight::GetProperty - Retrieves the value of a property or group of properties for the light.

Obj->[IDLgrLight::]GetProperty [, *PROPERTY*=*variable*]

IDLgrLight::Init - Initializes the light object.

Obj = OBJ_NEW('IDLgrLight'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrLight::]Init([*PROPERTY*=*value*])

IDLgrLight::SetProperty - Sets the value of a property or group of properties for the light.

Obj->[IDLgrLight::]SetProperty[, *PROPERTY*=*value*]

IDLgrModel - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated).

Properties: [, ACTIVE_POSITION{Get, Init, Set}=*integer*] [, ALL{Get}=*variable*] [, /HIDE{Get, Init, Set}] [, LIGHTING{Get, Init, Set}={0 | 1 | 2}] [, PARENT{Get}=*variable*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, RENDER_METHOD{Get, Init, Set}=*integer*] [, /SELECT_TARGET{Get, Init, Set}] [, TRANSFORM{Get, Init, Set}=*4x4 transformation matrix*]

IDLgrModel::Add - Adds a child to this Model.

Obj->[IDLgrModel::]Add, *Object* [, /ALIAS] [, POSITION=*index*]

IDLgrModel::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrModel::]Cleanup

IDLgrModel::Draw - Draws the specified picture to the specified graphics destination. *This method is provided for purposes of sub-classing only, and is intended to be called only from the Draw method of a subclass of IDLgrModel.*

Obj->[IDLgrModel::]Draw, *Destination*, *Picture*

IDLgrModel::GetByName - Finds contained objects by name and returns the object reference to the named object.

Result = *Obj*->[IDLgrModel::]GetByName(*Name*)

IDLgrModel::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object

Result = *Obj*->[IDLgrModel::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrModel::GetProperty - Retrieves the value of a property or group of properties for the model.

Obj->[IDLgrModel::]GetProperty [, *PROPERTY*=*variable*]

IDLgrModel::Init - Initializes the model object.

Obj = OBJ_NEW('IDLgrModel'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrModel::]Init([*PROPERTY*=*value*])

IDLgrModel::Reset - Sets the current transform matrix for the model object to the identity matrix.

Obj->[IDLgrModel::]Reset

IDLgrModel::Rotate - Rotates the model about the specified axis by the specified angle.

Obj->[IDLgrModel::]Rotate, *Axis*, *Angle* [, /PREMULTIPLY]

IDLgrModel::Scale - Scales model by the specified scaling factors.

Obj->[IDLgrModel::]Scale, *Sx*, *Sy*, *Sz* [, /PREMULTIPLY]

IDLgrModel::SetProperty - Sets the value of a property or group of properties for the model.

Obj->[IDLgrModel::]SetProperty[, *PROPERTY*=*value*]

IDLgrModel::Translate - Translates model by specified offsets.

Obj->[IDLgrModel::]Translate, *Tx*, *Ty*, *Tz* [, /PREMULTIPLY]

IDLgrMPEG - Creates an MPEG movie from an array of image frames.

Properties: [, ALL{Get}=*variable*] [, BITRATE{Get, Init, Set}=*value*] [, DIMENSIONS{Get, Init, Set}=*2-element array*] [, FILENAME{Get, Init, Set}=*string*] [, FORMAT{Get, Init, Set}={0 | 1}] [, FRAME_RATE {Get, Init, Set} ={1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, IFRAME_GAP{Get, Init, Set}=*integer value*] [, /INTERLACED{Get, Init, Set}] [, MOTION_VEC_LENGTH{Get, Init, Set}={1 | 2 | 3}] [, QUALITY{Get, Init, Set}=*value*{0 to100}] [, SCALE{Get, Init, Set}=*[xscale, yscale]*] [, /STATISTICS{Get, Init, Set}] [, TEMP_DIRECTORY{Init}=*string*]

IDLgrMPEG::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrMPEG::]Cleanup

IDLgrMPEG::GetProperty - Retrieves the value of a property or group of properties for the MPEG object.

Obj->[IDLgrMPEG::]GetProperty [, *PROPERTY*=*variable*]

IDLgrMPEG::Init - Initializes the MPEG object.

Obj = OBJ_NEW('IDLgrMPEG'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrMPEG::]Init([*PROPERTY*=*value*])

IDLgrMPEG::Put - Puts a given image into the MPEG sequence at the specified frame.

Obj->[IDLgrMPEG::]Put, *Image*[, *Frame*]

IDLgrMPEG::Save - Encodes and saves an MPEG sequence to a file.

Obj->[IDLgrMPEG::]Save [, FILENAME=*string*]

IDLgrMPEG::SetProperty - Sets the value of a property or group of properties for the MPEG object.

Obj->[IDLgrMPEG::]SetProperty[, *PROPERTY*=*value*]

IDLgrPalette - Represents a color lookup table that maps indices to red, green, and blue values.

Properties: [, ALL{Get}=*variable*] [, BLUE_VALUES{Get, Init, Set}=*vector*] [, BOTTOM_STRETCH{Get, Init, Set}=*value*{0 to 100}] [, GAMMA{Get, Init, Set}=*value*{0.1 to 10.0}] [, GREEN_VALUES{Get, Init, Set}=*vector*] [, N_COLORS{Get}=*variable*] [, NAME{Get, Init, Set}=string] [, RED_VALUES{Get, Init, Set}=*vector*] [, TOP_STRETCH{Get, Init, Set}=*value*{0 to 100}] [, UVALUE{Get, Init, Set}=*value*]

IDLgrPalette::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrPalette::]Cleanup

IDLgrPalette::GetRGB - Returns the RGB values contained in the palette at the given index.

Result = *Obj*->[IDLgrPalette::]GetRGB(*Index*)

IDLgrPalette::GetProperty - Retrieves the value of a property or group of properties for the palette.

Obj->[IDLgrPalette::]GetProperty [, *PROPERTY*=*variable*]

IDLgrPalette::Init - Initializes a palette object.

Obj = OBJ_NEW('IDLgrPalette', *aRed*, *aGreen*, *aBlue* [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLgrPalette::]Init([*aRed*, *aGreen*, *aBlue*] [, *PROPERTY*=*value*])

IDLgrPalette::LoadCT - Loads one of the IDL predefined color tables into an IDLgrPalette object.

Obj->[IDLgrPalette::]LoadCT, *TableNum* [, FILENAME=*colortable filename*]

IDLgrPalette::NearestColor - Returns the index of the color in the palette that best matches the given RGB values.

Result = *Obj*->[IDLgrPalette::]NearestColor(*Red*, *Green*, *Blue*)

IDLgrPalette::SetRGB - Sets the color values at a specified index in the palette to the specified Red, Green and Blue values.

Obj->[IDLgrPalette::]SetRGB, *Index*, *Red*, *Green*, *Blue*

IDLgrPalette::SetProperty - Sets the value of a property or group of properties for the palette.

Obj->[IDLgrPalette::]SetProperty[, *PROPERTY*=*value*]

IDLgrPattern - Describes which pixels are filled and which are left blank when an area is filled.

Properties: [, ALL{Get}=*variable*] [, NAME{Get, Init, Set}=string] [, ORIENTATION{Get, Init, Set}=*ccw degrees from horiz*] [, PATTERN{Get, Init, Set}=*32 x 32 bit array*] [, SPACING{Get, Init, Set}=*points*] [, STYLE{Get, Init, Set}={0 | 1 | 2}] [, THICK{Init}=*points*{1.0 to 10.0}] [, UVALUE{Get, Init, Set}=*value*]

IDLgrPattern::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrPattern::]Cleanup

IDLgrPattern::GetProperty - Retrieves the value of a property or group of properties for the pattern.

Obj->[IDLgrPattern::]GetProperty [, *PROPERTY*=*variable*]

IDLgrPattern::Init - Initializes the pattern object.

Obj = OBJ_NEW('IDLgrPattern'[, *Style*] [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLgrPattern::]Init([*Style*] [, *PROPERTY*=*value*])

IDLgrPattern:SetProperty - Sets the value of a property or group of properties for the pattern.

Obj->[IDLgrPattern::]SetProperty[, *PROPERTY*=*value*]

IDLgrPlot - Creates set of polylines connecting data points in 2D space.

Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB vector* | , VERT_COLORS{Get, Init, Set}=*vector*] [, DATA{Get}=*variable*] [, DATAX{Init, Set}=*vector*] [, DATAY{Init, Set}=*vector*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DOUBLE{Get, Init, Set}=*value*]

IDLgrPlot Properties - *continued*

[, /HIDE{Get, Init, Set}] [, /HISTOGRAM{Get, Set}] [, LINESTYLE{Get, Init, Set}=*integer or two-element vector*] [, MAX_VALUE{Get, Init, Set}=*value*] [, MIN_VALUE{Get, Init, Set}=*value*] [, NSUM{Get, Init, Set}=*value*] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT{Get}=*variable*] [, /POLAR{Get, Init, Set}] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, /RESET_DATA{Init, Set}] [, SHARE_DATA{Init, Set}=*objref*] [, SYMBOL{Get, Init, Set}=*objref(s)*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, /USE_ZVALUE{Init}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get, Init, Set}=*[xmin, xmax]*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get, Init, Set}=*[ymin, ymax]*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*] [, ZVALUE{Get, Init, Set}=*value*]

IDLgrPlot::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrPlot::]Cleanup

IDLgrPlot::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj*->[IDLgrPlot::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrPlot::GetProperty - Retrieves the value of the property or group of properties for the plot.

Obj->[IDLgrPlot::]GetProperty[, *PROPERTY*=*variable*]

IDLgrPlot::Init - Initializes the plot object.

Obj = OBJ_NEW('IDLgrPlot'[, [*X*,] *Y*] [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLgrPlot::]Init([[*X*,] *Y*] [, *PROPERTY*=*value*])

IDLgrPlot::SetProperty - Sets the value of the property or group of properties for the plot.

Obj->[IDLgrPlot::]SetProperty[, *PROPERTY*=*value*]

IDLgrPolygon - Represents one or more polygons that share a set of vertices and rendering attributes.

Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, AMBIENT{Get, Init, Set}=*RGB vector*] [, BOTTOM{Get, Init, Set}=*index or RGB vector*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB vector* | , VERT_COLORS{Get, Init, Set}=*vector*] [, DATA{Get, Init, Set}=*array*] [, DEPTH_OFFSET{Get, Init, Set}=*value*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DIFFUSE{Get, Init, Set}=*RGB vector*] [, DOUBLE{Get, Init, Set}=*value*] [, EMMISION{Get, Init, Set}=*RGB vector*]

IDLgrPolygon Properties - *continued*
[, FILL_PATTERN{Get, Init, Set}=*objref to IDLgrPattern object*] [, /HIDDEN_LINES{Init}] [, /HIDE{Get, Init, Set}] [, LINESTYLE{Get, Init, Set}=*value*] [, NAME{Get, Init, Set}=*string*] [, NORMALS{Get, Init, Set}=*array*] [, PALETTE{Init}=*objref*] [, PARENT{Get}=*variable*] [, POLYGONS{Get, Init, Set}=*array of polygon descriptions*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, REJECT{Get, Init, Set}={0 | 1 | 2}] [, /RESET_DATA{Init, Set}] [, SHADE_RANGE{Get, Init, Set}=*array*] [, SHADING{Get, Init, Set}={0 | 1}] [, SHARE_DATA{Init, Set}=*objref*] [, SHININESS{Get, Init, Set}=*value*] [, SPECULAR{Get, Init, Set}= *RGB vector*] [, STYLE{Get, Init, Set}={0 | 1 | 2}] [, TEXTURE_COORD{Get, Init, Set}=*array*] [, /TEXTURE_INTERP{Get, Init, Set}] [, TEXTURE_MAP{Get,Init, Set}=*objref to IDLgrImage object*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZERO_OPACITY_SKIP{Get, Init, Set}={0 | 1}] [, ZRANGE{Get}=*variable*]

IDLgrPolygon::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj->*[IDLgrPolygon::]Cleanup

IDLgrPolygon::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.
Result = *Obj->*[IDLgrPolygon::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrPolygon::GetProperty - Retrieves the value of the property or group of properties for the polygons.
Obj->[IDLgrPolygon::]GetProperty [, *PROPERTY=variable*]

IDLgrPolygon::Init - Initializes the polygons object.
Obj = OBJ_NEW('IDLgrPolygon'[, *X* [, *Y*[, *Z*]]] [, *PROPERTY=value*]) or
Result = *Obj->*[IDLgrPolygon::]Init([*X*, [*Y*, [*Z*]]] [, *PROPERTY=value*])

IDLgrPolygon::SetProperty - Sets the value of the property or group of properties for the polygons.
Obj->[IDLgrPolygon::]SetProperty [, *PROPERTY=value*]

IDLgrPolyline - Represents one or more polylines that share a set of vertices and rendering attributes.
Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB vector* | , VERT_COLORS{Get, Init, Set}=*vector*] [, DATA{Get, Init, Set}=*array*]

IDLgrPolyline Properties - *continued*
[, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DOUBLE{Get, Init, Set}=*value*] [, /HIDE{Get, Init, Set}] [, LABEL_NOGAPS{Get, Init, Set}=*vector*] [, LABEL_OBJECTS{Get, Init, Set}=*objref*] [, LABEL_OFFSETS{Get, Init, Set}=*scalar or vector of offsets*}] [, LABEL_POLYLINES{Get, Init, Set}=*scalar or vector of polyline indices*] [, LABEL_USE_VERTEX_COLOR{Get, Init, Set}=*value*] [, LINESTYLE{Get, Init, Set}=*value*] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT{Get}=*variable*] [, POLYLINES{Get, Init, Set}=*array of polyline descriptions*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, /RESET_DATA{Init, Set}] [, SHADING{Get, Init, Set}={0 | 1}] [, SHARE_DATA{Init, Set}=*objref*] [, SYMBOL{Get, Init, Set}=*objref(s)*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, USE_LABEL_COLOR{Get, Init, Set}=*vector*] [, USE_LABEL_ORIENTATION{Get, Init, Set}=*vector*] [, /USE_TEXT_ALIGNMENTS{Get, Init, Set}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrPolyline::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj->*[IDLgrPolyline::]Cleanup

IDLgrPolyline::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.
Result = *Obj->*[IDLgrPolyline::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrPolyline::GetProperty - Retrieves the value of a property or group of properties for the polylines.
Obj->[IDLgrPolyline::]GetProperty [, *PROPERTY=variable*]

IDLgrPolyline::Init - Initializes the polylines object.
Obj = OBJ_NEW('IDLgrPolyline' [, *X* [, *Y*[, *Z*]]] [, *PROPERTY=value*]) or
Result = *Obj->*[IDLgrPolyline::]Init([*X*, [*Y*, [*Z*]]] [, *PROPERTY=value*])

IDLgrPolyline::SetProperty - Sets the value of a property or group of properties for the polylines.
Obj->[IDLgrPolyline::]SetProperty [, *PROPERTY=value*]

IDLgrPrinter - Represents a hardcopy graphics destination.

Properties: [, ALL{Get}=*variable*]
[, COLOR_MODEL{Get, Init}={0 | 1}]
[, DIMENSIONS{Get}=*variable*]
[, GRAPHICS_TREE{Get, Init, Set}=*objref of type IDLgrScene, IDLgrViewgroup, or IDLgrView*]
[, /LANDSCAPE{Get, Init, Set}] [, N_COLORS{Get, Init}=*integer*{2 to 256}] [, N_COPIES{Get, Init, Set}=*integer*] [, NAME{Get, Init, Set}=*string*]
[, PALETTE{Get, Init, Set}=*objref*]
[, PRINT_QUALITY{Get, Init, Set}={0 | 1 | 2}]
[PRINTER_NAME{Get}=*variable*] [, QUALITY{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, RESOLUTION{Get}=*variable*]
[, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]

IDLgrPrinter::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrPrinter::]Cleanup

IDLgrPrinter::Draw - Draws a picture to this graphics destination.

Obj->[IDLgrPrinter::]Draw [, *Picture*]
[, VECT_SORTING={ 0 | 1 }]
[, VECT_TEXT_RENDER_METHOD={ 0 | 1 }]
[, VECTOR={ 0 | 1 }]

IDLgrPrinter::GetContiguousPixels - Returns an array of long integers whose length is equal to the number of colors available in the index color mode (value of N_COLORS property).

Return = *Obj*->[IDLgrPrinter::]GetContiguousPixels( )

IDLgrPrinter::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj*->[IDLgrPrinter::]GetFontnames(
FamilyName [, IDL_FONTS={0 | 1 | 2}]
[, STYLES=*string*])

IDLgrPrinter::GetProperty - Retrieves the value of a property or group of properties for the printer.

Obj->[IDLgrPrinter::]GetProperty
[, *PROPERTY*=*variable*]

IDLgrPrinter::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered on the printer.

Result = *Obj*->[IDLgrPrinter::]GetTextDimensions(
TextObj [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrPrinter::Init - Initializes the printer object.

Obj = OBJ_NEW('IDLgrPrinter'[, *PROPERTY*=*value*]) or
Result = *Obj*->[IDLgrPrinter::]Init([*PROPERTY*=*value*])

IDLgrPrinter::NewDocument - Closes current document (page or group of pages), which causes pending output to be sent to the printer, finishing the printer job.

Obj->[IDLgrPrinter::]NewDocument

IDLgrPrinter::NewPage - Issues new page command to printer.

Obj->[IDLgrPrinter::]NewPage

IDLgrPrinter::QueryRequiredTiles - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

Result = *Obj*->[IDLgrWindow::]QueryRequiredTiles
(*View, Image* [, COUNT=*variable*]
[, ALL_VISIBLE=*value*])

IDLgrPrinter::SetProperty - Sets the value of a property or group of properties for the printer.

Obj->[IDLgrPrinter::]SetProperty[, *PROPERTY*=*value*]

IDLgrROI - Object graphics representation of a region of interest.

Properties: [, ALL{Get}=*variable*]
[,ALPHA_CHANNEL{Get, Init, Set}=*value*]
[, CLIP_PLANES{Get, Init, Set}=*array*]
[, COLOR{Get, Init, Set}=*vector*]
[, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}]
[, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DOUBLE{Get, Init, Set}=*value*]
[, /HIDE{Get, Init, Set}] [, LINESTYLE{Get, Init, Set}=*value*] [, PALETTE{Get, Init, Set}=*objref*]
[, /REGISTER_PROPERTIES{Get, Init, Set}]
[, STYLE{Get, Init, Set}={ 0 | 1 | 2 }]
[, SYMBOL{Get, Init, Set}=*objref*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, XCOORD_CONV{Get, Init, Set}=*[s_0, s_1]*] [, XRANGE{Get}=*variable*]
[, YCOORD_CONV{Get, Init, Set}=*[s_0, s_1]*]
[, YRANGE{Get}=*variable*]
[, ZCOORD_CONV{Get, Init, Set}=*[s_0, s_1]*] [, ZRANGE{Get}=*variable*]

IDLgrROI::Cleanup - Performs all cleanup for the object.

OBJ_DESTROY, *Obj* or *Obj* –> [IDLgrROI::]Cleanup

IDLgrROI::GetProperty - Retrieves the value of a property or group of properties for the Object Graphics region.

Obj –> [IDLgrROI::]GetProperty[, *PROPERTY*=*variable*]

IDLgrROI::Init - Initializes an Object Graphics region of interest.

Obj = OBJ_NEW('IDLgrROI' [, *X*[, *Y*[, *Z*]]]
[, *PROPERTY*=*value*]) or
Result = *Obj* –> [IDLgrROI::]Init([*X*[, *Y*[, *Z*]]]
[, *PROPERTY*=*value*])

IDLgrROI::PickVertex - Picks a vertex of the region that, when projected onto the given destination device, is nearest to the given 2D device coordinate.

Result = *Obj* –> [IDLgrROI::]PickVertex(*Dest, View,*
Point [, PATH=*objref*])

IDLgrROI::SetProperty - Sets the value of a property or group of properties for the Object Graphics region.

Obj –> [IDLgrROI::]SetProperty[, *PROPERTY*=*value*]

IDLgrROIGroup - Object Graphics representation of a group of regions of interest.

Properties: [, ALL{Get}=*variable*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, /HIDE{Get, Init, Set}] [, PARENT{Get}=*variable*] [, XCOORD_CONV{Get, Init, Set}=*[s_0, s_1]*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*[s_0, s_1]*] [, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*[s_0, s_1]*] [, ZRANGE{Get}=*variable*]

IDLgrROIGroup::Add - Adds a region to the region group.

Obj –> [IDLgrROIGroup::]Add, *ROI*

IDLgrROIGroup::Cleanup - Performs all cleanup for the object.

OBJ_DESTROY, *Obj* or *Obj* –> [IDLgrROIGroup::]Cleanup

IDLgrROIGroup::GetProperty - Retrieves the value of a property or group of properties for the region group.

Obj –> [IDLgrROIGroup::]GetProperty [, *PROPERTY*=*variable*]

IDLgrROIGroup::Init - Initializes an Object Graphics region of interest group object.

Obj = OBJ_NEW('IDLgrROIGroup'[, *PROPERTY*=*value*]) or *Result* = *Obj* –> [IDLgrROIGroup::]Init([, *PROPERTY*=*value*])

IDLgrROIGroup::PickRegion - Picks a region within the group that, when projected onto the given destination device, is nearest to the given 2D device coordinate.

Result = *Obj* –> [IDLgrROIGroup::]PickRegion(*Dest*, *View*, *Point* [, PATH=*objref*])

IDLgrROIGroup::SetProperty - Sets the value of a property or group of properties for the region group.

Obj –> [IDLgrROIGroup::]SetProperty [, *PROPERTY*=*value*]

IDLgrScene - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects.

Properties: [, ALL{Get}=*variable*] [, COLOR{Get, Init, Set}=*index or RGB vector*] [, /HIDE{Get, Init, Set}] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, /TRANSPARENT{Get, Init, Set}]

IDLgrScene::Add - Verifies that the added item is an instance of an IDLgrView or IDLgrViewgroup object.

Obj->[IDLgrScene::]Add, *View* [, POSITION=*index*]

IDLgrScene::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrScene::]Cleanup

IDLgrScene::GetByName - Finds contained objects by name and returns the object reference to the named object.

Result = *Obj*->[IDLgrScene::]GetByName(*Name*)

IDLgrScene::GetProperty - Retrieves the value of a property or group of properties for the contour.

Obj->[IDLgrScene::]GetProperty [, *PROPERTY*=*variable*]

IDLgrScene::Init - Initializes the scene object.

Obj = OBJ_NEW('IDLgrScene'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrScene::]Init([, *PROPERTY*=*value*])

IDLgrScene::SetProperty - Sets the value of one or more properties for the scene.

Obj->[IDLgrScene::]SetProperty[, *PROPERTY*=*value*]

IDLgrSurface - A shaded or vector representation of a mesh grid.

Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, AMBIENT{Get, Init, Set}=*RGB vector*] [, PARENT{Get}=*variable*] [, XRANGE{Get}=*variable*] [, YRANGE{Get}=*variable*] [, ZRANGE{Get}=*variable*] [, BOTTOM{Get, Set}=*index or RGB vector*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB vector*] [, DATAX{Init, Set}=*vector or 2D array*] [, DATAY{Init, Set}=*vector or 2D array*] [, DATAZ{Init, Set}=*2D array*] [, DEPTH_OFFSET{Get, Init, Set}=*value*] [, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DIFFUSE{Get, Init, Set}=*RGB vector*] [, DOUBLE{Get, Init, Set}=*value*] [, EMMISION{Get, Init, Set}=*RGB vector*] [, /EXTENDED_LEGO{Get, Init, Set}] [, /HIDDEN_LINES{Get, Init, Set}] [, /HIDE{Get, Init, Set}] [, LINESTYLE{Get, Init, Set}=*value*] [, MAX_VALUE{Get, Init, Set}=*value*] [, MIN_VALUE{Get, Init, Set}=*value*] [, PALETTE{Get, Init, Set}=*objref*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, /RESET_DATA{Init, Set}] [, SHADE_RANGE{Get, Init, Set}=*[index of darkest pixel, index of brightest pixel]*] [, SHADING{Get, Init, Set}={0 | 1}] [, SHARE_DATA{Init, Set}=*objref*] [, SHININESS{Get, Init, Set}=*value*] [, /SHOW_SKIRT{Get, Init, Set}] [, SKIRT{Get, Init, Set}=*Z value*] [, SPECULAR{Get, Init, Set}=*RGB vector*] [, STYLE{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, TEXTURE_COORD{Get, Init, Set}=*array*] [, /TEXTURE_HIGHRES{Get, Init, Set}] [, /TEXTURE_INTERP{Get, Init, Set}] [, TEXTURE_MAP{Get, Init, Set}=*objref to IDLgrImage*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, /USE_TRIANGLES{Get, Init, Set}]

IDLgrSurface Properties - *continued*
[, VERT_COLORS{Get, Init, Set}=*vector*]
[, XCOORD_CONV{Get, Init, Set}=*vector*]
[, YCOORD_CONV{Get, Init, Set}=*vector*]
[, ZCOORD_CONV{Get, Init, Set}=*vector*]
[, ZERO_OPACITY_SKIP{Get, Init, Set}={0 | 1}]

IDLgrSurface::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDLgrSurface::]Cleanup

IDLgrSurface::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.
Result = *Obj*->[IDLgrSurface::]GetCTM(
[, DESTINATION=*objref*] [, PATH=*objref(s)*]
[, TOP=*objref to IDLgrModel object*])

IDLgrSurface::GetProperty - Retrieves the value of a property or group of properties for the surface.
Obj->[IDLgrSurface::]GetProperty
[, *PROPERTY*=*variable*]

IDLgrSurface::Init - Initializes the surface object.
Obj = OBJ_NEW('IDLgrSurface' [, *Z* [, *X*, *Y*]]
[, *PROPERTY*=*value*]) or
Result = *Obj*->[IDLgrSurface::]Init([*Z* [, *X*, *Y*]]
[, *PROPERTY*=*value*])

IDLgrSurface::SetProperty - Sets the value of a property or group of properties for the surface.
Obj->[IDLgrSurface::]SetProperty[, *PROPERTY*=*value*]

IDLgrSymbol - Represents a graphical element that is plotted relative to a particular position.
Properties: [, ALL{Get}=*variable*]
[,ALPHA_CHANNEL{Get, Init, Set}=*value*]
[, COLOR{Get, Init, Set}=*index or RGB vector*]
[, DATA{Get, Init, Set}=*integer or objref*]
[, NAME{Get, Init, Set}=string] [, SIZE{Get, Init, Set}=*vector*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, UVALUE{Get, Init, Set}=*value*]

IDLgrSymbol::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDLgrSymbol::]Cleanup

IDLgrSymbol::GetProperty - Retrieves the value of a property or group of properties for the symbol.
Obj->[IDLgrSymbol::]GetProperty
[, *PROPERTY*=*variable*]

IDLgrSymbol::Init - Initializes the plot symbol.
Obj = OBJ_NEW('IDLgrSymbol'[, *Data*]
[, *PROPERTY*=*value*]) or
Result = *Obj*->[IDLgrSymbol::]Init([*Data*]
[, *PROPERTY*=*value*])

IDLgrSymbol::SetProperty - Sets the value of a property or group of properties for the symbol.
Obj->[IDLgrSymbol::]SetProperty[, *PROPERTY*=*value*]

IDLgrTessellator - Converts a simple concave polygon (or a simple polygon with "holes") into a number of simple convex polygons (general triangles).

IDLgrTessellator::AddPolygon - Adds a polygon to the tessellator object.
Obj->[IDLgrTessellator::]AddPolygon, *X* [, *Y*[, *Z*]]
[, AUXDATA=*array of auxiliary data*] [, /INTERIOR]
[, POLYGON=*array of polygon descriptions*]

IDLgrTessellator::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY,*Obj* or
Obj->[IDLgrTessellator::]Cleanup

IDLgrTessellator::Init - Initializes the tessellator object.
Obj = OBJ_NEW('IDLgrTessellator') or
Result = *Obj*->[IDLgrTessellator::]Init()

IDLgrTessellator::Reset - Resets the object's internal state.
Obj->[IDLgrTessellator::]Reset

IDLgrTessellator::Tessellate - Performs the actual tessellation.
Result = *Obj*->[IDLgrTessellator::]Tessellate(*Vertices*, *Poly* [, AUXDATA=*variable*] [, /QUIET])

IDLgrText - Represents one or more text strings that share common rendering attributes.
Properties: [, ALL{Get}=*variable*] [, ALIGNMENT{Get, Init, Set}=*value*{0.0 to 1.0}]
[, ALPHA_CHANNEL{Get, Init, Set}=*value*{0.0 to 1.0}] [, BASELINE{Get, Init, Set}=*vector*]
[, CHAR_DIMENSIONS{Get, Init, Set}=*[width, height]*]
[, CLIP_PLANES{Get, Init, Set}=*array*]
[, COLOR{Get, Init, Set}=*index or RGB vector*]
[, DEPTH_TEST_DISABLE{Get, Init, Set}={0 | 1 | 2}]
[, DEPTH_TEST_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH_WRITE_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, /ENABLE_FORMATTING{Get, Init, Set}] [, /FILL_BACKGROUND{Get, Init, Set}]
[, FILL_COLOR{Get, Init, Set}=*index or RGB vector*]
[, FONT{Get, Init, Set}=*objref*] [, /HIDE{Get, Init, Set}]
[, /KERNING{Get, Init, Set}] [, LOCATIONS{Get, Init, Set}=*array*] [, NAME{Get, Init, Set}=*string*]
[, /ONGLASS{Get, Init, Set}] [, PALETTE{Get, Init, Set}=*objref*] [, PARENT{Get}=*variable*]
[, RECOMPUTE_DIMENSIONS{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER_PROPERTIES{Get, Init, Set}]
[, RENDER_METHOD{Get, Init, Set}={0 | 1}]
[, STRINGS{Get, Init, Set}=*string or vector of strings*]
[, UPDIR{Get, Init, Set}=*vector*] [, UVALUE{Get, Init, Set}=*value*] [, VERTICAL_ALIGNMENT{Get, Init, Set}=*value*{0.0 to 1.0}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*]
[, YCOORD_CONV{Get, Init, Set}=*vector*]
[, YRANGE{Get}=*variable*] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]

IDLgrText::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDLgrText::]Cleanup

IDLgrText::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj*->[IDLgrText::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrText::GetProperty - Retrieves the value of a property or group of properties for the text.

Obj->[IDLgrText::]GetProperty[, *PROPERTY*=*variable*]

IDLgrText::Init - Initializes the text object.

Obj = OBJ_NEW('IDLgrText'[, *String/string array*] [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLgrText::]Init(*String/sting array*] [, *PROPERTY*=*value*])

IDLgrText::SetProperty - Sets the value of a property or group of properties for the text.

Obj->[IDLgrText::]SetProperty[, *PROPERTY*=*value*]

IDLgrView - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class.

Properties: [, ALL{Get}=*variable*] [, PARENT{Get}=*variable*] [, COLOR{Get, Init, Set}=*index or RGB vector*] [, DEPTH_CUE{Get, Init, Set}=*[zbright, zdim]*] [, DIMENSIONS{Get, Init, Set}=*[width, height]*] [, DOUBLE {Get, Init, Set}=*value*] [, EYE{Get, Init, Set}=*distance*] [, LOCATION{Get, Init, Set}=*[x, y]*] [, PROJECTION{Get, Init, Set}={1 | 2}] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, /TRANSPARENT{Get, Init, Set}] [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}] [, VIEWPLANE_RECT{Get, Init, Set}=*[x, y, width, height]*] [, ZCLIP{Get, Init, Set}=*[near, far]*]

IDLgrView::Add - Adds a child to this view.

Obj->[IDLgrView::]Add, *Model* [, POSITION=*index*]

IDLgrView::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLgrView::]Cleanup

IDLgrView::GetByName - Finds contained objects by name.

Result = *Obj*->[IDLgrView::]GetByName(*Name*)

IDLgrView::GetProperty - Retrieves the value of the property or group of properties for the view.

Obj->[IDLgrView::]GetProperty [, *PROPERTY*=*variable*]

IDLgrView::Init - Initializes the view object.

Obj = OBJ_NEW('IDLgrView'[, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLgrView::]Init([*PROPERTY*=*value*])

IDLgrView::SetProperty - Sets the value of the property or group of properties for the view.

Obj->[IDLgrView::]SetProperty[, *PROPERTY*=*value*]

IDLgrViewgroup - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects.

Properties: [, ALL{Get}=*variable*] [, /HIDE{Get, Init, Set}] [, PARENT{Get}=*variable*] [, /REGISTER_PROPERTIES{Get, Init, Set}]

IDLgrViewgroup::Add - Verifies that the added item is not an instance of the IDLgrScene or IDLgrViewgroup object.

Obj->[IDLgrViewgroup::]Add, *Object* [, POSITION=*index*]

IDLgrViewgroup::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj* or *Obj*->[IDLgrViewgroup::]Cleanup

IDLgrViewgroup::GetByName - Finds contained objects by name.

Result = *Obj*->[IDLgrViewgroup::]GetByName(*Name*)

IDLgrViewgroup::GetProperty - Retrieves the value of a property or group of properties for the viewgroup object.

Obj->[IDLgrViewgroup::]GetProperty [, *PROPERTY*=*variable*]

IDLgrViewgroup::Init - Initializes the viewgroup object.

Obj = OBJ_NEW('IDLgrViewgroup'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLgrViewgroup::]Init([*PROPERTY*=*value*])

IDLgrViewgroup::SetProperty - Sets the value of a property or group of properties for the viewgroup.

Obj->[IDLgrViewgroup::]SetProperty [, *PROPERTY*=*value*]

IDLgrVolume - Represents mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions.

Properties: [, ALL{Get}=*variable*] [,ALPHA_CHANNEL{Get, Init, Set}=*value*] [, AMBIENT{Get, Init, Set}=*RGB vector*] [, BOUNDS{Get, Init, Set}=*[xmin, ymin, zmin, xmax, ymax, zmax]*] [, CLIP_PLANES{Get, Init, Set}=*array*] [, COMPOSITE_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3}] [, DATA0{Get, Init, Set}=*[d_x, d_y, d_z]*] [, DATA1{Get, Init, Set}=*[d_x, d_y, d_z]*] [, DATA2{Get, Init, Set}=*[d_x, d_y, d_z]*] [, DATA3{Get, Init, Set}=*[d_x, d_y, d_z]*] [, DEPTH_CUE{Get, Init, Set}=*[zbright, zdim]*] [, /HIDE{Get, Init, Set}] [, HINTS{Get, Init, Set}={0 | 1 | 2 | 3}] [, /INTERPOLATE{Get, Init, Set}] [, /LIGHTING_MODEL{Get, Init, Set}] [, OPACITY_TABLE0{Get, Init, Set}=*256-element byte array*] [, OPACITY_TABLE1{Get, Init, Set}=*256-element byte array*] [, PARENT{Get}=*variable*] [, RENDER_STEP{Get, Init, Set}=*[x, y, z]*]

IDLgrVolume Properties - *continued*
[, RGB_TABLE0{Get, Init, Set}=*256 x 3-element byte array*] [, RGB_TABLE1{Get, Init, Set}=*256 x 3-element byte array*] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, /TWO_SIDED{Get, Init, Set}] [, VALID_DATA{Get}=*variable*] [, VOLUME_SELECT{Get, Init, Set}={0 | 1 | 2}] [, XCOORD_CONV{Get, Init, Set}=*vector*] [, XRANGE{Get}=*variable*] [, YCOORD_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*] [, /ZBUFFER{Get, Init, Set}] [, ZCOORD_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*] [, ZERO_OPACITY_SKIP{Get, Init, Set}={0 | 1}]

IDLgrVolume::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDLgrVolume::]Cleanup

IDLgrVolume::ComputeBounds - Computes the smallest bounding box that contains all voxels whose opacity lookup is greater than a given opacity value.
Obj->[IDLgrVolume::]ComputeBounds [, OPACITY=*value*] [, /RESET] [, VOLUMES=*int array*]

IDLgrVolume::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.
Result = *Obj*->[IDLgrVolume::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrVolume::GetProperty - Retrieves the value of a property or group of properties for the volume.
Obj->[IDLgrVolume::]GetProperty [, *PROPERTY*=*variable*] [, /NO_COPY]

IDLgrVolume::Init - Initializes the volume object.
Obj = OBJ_NEW('IDLgrVolume' [, *vol0* [, *vol1* [, *vol2* [, *vol3*]]]] [, *PROPERTY*=*value*] [, /NO_COPY]) or
Result = *Obj*->[IDLgrVolume::]Init([*vol0* [, *vol1* [, *vol2* [, *vol3*]]]] [, *PROPERTY*=*value*] [, /NO_COPY])

IDLgrVolume::PickVoxel - Computes the coordinates of the voxel projected to a location specified by the 2D device coordinates point, [x_i, y_i], and the current Z-buffer.
Result = *Obj*->[IDLgrVolume::]PickVoxel (*Win*, *View*, *Point* [, PATH=*objref(s)*])

IDLgrVolume::SetProperty - Sets the value of a property or group of properties for the volume.
Obj->[IDLgrVolume::]SetProperty[, *PROPERTY*=*value*] [, /NO_COPY]

IDLgrVRML - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.
Properties: [, ALL{Get}=*variable*] [, SCREEN_DIMENSIONS{Get}=*variable*] [, COLOR_MODEL{Get, Init}={0 | 1}] [, DIMENSIONS{Get, Init, Set}=*[width, height]*] [, FILENAME{Get, Init, Set}=*string*] [, GRAPHICS_TREE{Get, Init, Set}=*objref*] [, N_COLORS{Get, Init}=*integer*{2 to 256}] [, PALETTE{Get, Init, Set}=*objref*] [, QUALITY{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER_PROPERTIES{Get, Init, Set}] [, RESOLUTION{Get, Init, Set}=*[xres, yres]*] [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}] [, WORLDINFO{Init}=*string array*] [, WORLDTITLE{Init}=*string*]

IDLgrVRML::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj*->[IDLgrVRML::]Cleanup

IDLgrVRML::Draw - Draws a picture to this graphics destination.
Obj->[IDLgrVRML::]Draw [, *Picture*]

IDLgrVRML::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.
Obj–>[IDLgrVRML::]GetDeviceInfo [, ALL=*variable*] [, MAX_NUM_CLIP_PLANES=*variable*] [, MAX_TEXTURE_DIMENSIONS=*variable*] [, MAX_VIEWPORT_DIMENSIONS=*variable*] [, NAME=*variable*] [, NUM_CPUS=*variable*] [, VENDOR=*variable*] [, VERSION=*variable*]

IDLgrVRML::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.
Return = *Obj*->[IDLgrVRML::]GetFontnames (*FamilyName* [, IDL_FONTS={0 | 1 | 2}] [, STYLES=*string*])

IDLgrVRML::GetProperty - Retrieves the value of a property or group of properties for the VRML object.
Obj->[IDLgrVRML::]GetProperty [, *PROPERTY*=*variable*]

IDLgrVRML::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the clipboard buffer.
Result = *Obj*->[IDLgrVRML::]GetTextDimensions(*TextObj* [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrVRML::Init - Initializes the VRML object.
Obj = OBJ_NEW('IDLgrVRML'[, *PROPERTY*=*value*]) or
Result = *Obj*->[IDLgrVRML::]Init([*PROPERTY*=*value*])

IDLgrVRML::SetProperty - Sets the value of a property or group of properties for the VRML world.
Obj->[IDLgrVRML::]SetProperty[, *PROPERTY*=*value*]

IDLgrWindow - Represents an on-screen area on a display device that serves as a graphics destination.

Properties: [, ALL{Get}=*variable*]
[, COLOR_MODEL{Get, Init}={0 | 1}]
[, DIMENSIONS{Get, Init, Set} =*[width, height]*]
[, DISPLAY_NAME{Get}(X Windows Only)=*variable*]
[, IMAGE_DATA{Get}=*variable*]
[, GRAPHICS_TREE{Get, Init, Set}=*objref of type IDLgrScene, IDLgrViewgroup, or IDLgrView*]
[, LOCATION{Get, Init, Set}=*[x, y]*]
[, MINIMUM_VIRTUAL_DIMENSIONS{Get, Init, Set}=[*width, height*]] [, N_COLORS{Get, Init}=*integer*{2 to 256}] [, PALETTE{Get, Init, Set}=*objref*] [, QUALITY{Get, Init, Set}={0 | 1 | 2}]
[, /REGISTER_PROPERTIES{Get, Init, Set}]
[, RENDERER{Get, Init}={0 | 1}] [, RESOLUTION {Get}=*variable*] [, RETAIN{Get, Init}={0 | 1 | 2}]
[, SCREEN_DIMENSIONS{Get}=*variable*]
[, TITLE{Get, Init, Set}=*string*] [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]
[, VIEWPORT_DIMENSIONS{Get}=*[width, height]*]
[, VIRTUAL_DIMENSIONS{Get, Init, Set}=*[width, height]*] [, VISIBLE_LOCATION{Get, Init, Set}=*[x, y]*]
[, ZBUFFER_DATA{Get}=*variable*]
[, ZOOM_BASE{Get, Init, Set}=*value*]
[, ZOOM_NSTEP{Get}=*variable*]

IDLgrWindow::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj->*[IDLgrWindow::]Cleanup

IDLgrWindow::Draw - Draws the specified scene or view object to this graphics destination.

Obj->[IDLgrWindow::]Draw [, *Picture*]
[, CREATE_INSTANCE={1 | 2}]
[, DRAW_INSTANCE={1 | 2}]

IDLgrWindow::Erase - Erases the entire contents of the window.

Obj->[IDLgrWindow::]Erase [, COLOR=*index or RGB vector*]

IDLgrWindow::GetContiguousPixels - Returns an array of long integers whose length is equal to the number of colors available in the index color mode (value of N_COLORS property).

Return = Obj->[IDLgrWindow::]GetContiguousPixels( )

IDLgrWindow::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Obj –> [IDLgrWindow::]GetDeviceInfo [, ALL=*variable*]
[, MAX_NUM_CLIP_PLANES=*variable*]
[, MAX_TEXTURE_DIMENSIONS=*variable*]
[MAX_TILE_DIMENSIONS=*variable*]
[, MAX_VIEWPORT_DIMENSIONS=*variable*]
[, NAME=*variable*] [, NUM_CPUS=*variable*]
[, VENDOR=*variable*] [, VERSION=*variable*]

IDLgrWindow::GetDimensions - Returns a two-element vector, [width, height], representing the visible dimensions (in device units) of this window.

Result = Obj –> [IDLgrWindow::]GetDimensions([, MINIMUM_VIRTUAL_DIMENSIONS=*variable*]
[, ORIGINAL_VIRTUAL_DIMENSIONS=*variable*]
[, VIRTUAL_DIMENSIONS=*variable*]
[, VISIBLE_LOCATION=*variable*])

DLgrWindow::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = Obj->[IDLgrWindow::]GetFontnames(*FamilyName*
[, IDL_FONTS={0 | 1 | 2}] [, STYLES=*string*])

IDLgrWindow::GetProperty - Retrieves the value of a property or group of properties for the window.

Obj->[IDLgrWindow::]GetProperty
[, *PROPERTY=variable*]

IDLgrWindow::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the window.

Result = Obj->[IDLgrWindow::]GetTextDimensions(*TextObj* [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrWindow::Iconify - Iconifies or de-iconifies the window.

Obj->[IDLgrWindow::]Iconify, *IconFlag*

IDLgrWindow::Init - Initializes the window object.

Obj = OBJ_NEW('IDLgrWindow'[, *PROPERTY=value*])
or *Result = Obj->*[IDLgrWindow::]Init(
[, *PROPERTY=value*])

IDLgrWindow::Pickdata - Maps a point in the 2D device space of the window to a point in the 3D data space of an object tree.

Result = Obj->[IDLgrWindow::]Pickdata(*View, Object, Location, XYZLocation* [, DIMENSIONS=*[width,height]*]
[, PATH=*objref(s)*] [, PICK_STATUS=*variable*])

IDLgrWindow::QueryRequiredTiles - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

Result = Obj->[IDLgrWindow::]QueryRequiredTiles
(*View, Image* [, COUNT=*variable*]
[, ALL_VISIBLE=*value*])

IDLgrWindow::Read - Reads an image from a window.

Result = Obj->[IDLgrWindow::]Read( )

IDLgrWindow::Select - Returns a list of objects selected at a specified location.

Result = Obj->[IDLgrWindow::]Select(*Picture, XY*
[, DIMENSIONS=*[width, height]*] [, /ORDER]
[, SUB_SELECTION=*variable*] [, UNITS={0 | 1 | 2 | 3}])

IDLgrWindow::SetCurrentCursor - Sets the current cursor image to be used while positioned over a drawing area.

Obj->[IDLgrWindow::]SetCurrentCursor [, *CursorName*]
[, IMAGE=*16 x 16 bitmap*] [, MASK=*16 x 16 bitmap*]
[, HOTSPOT=*[x, y]*]
X Windows Only Keywords: [, STANDARD=*index*]

IDLgrWindow::SetCurrentZoom - Sets the current zoom factor for this window. The current zoom factor, the virtual canvas dimensions, and the visible dimensions of the window are updated to reflect the new zoom factor.

Obj->[IDLgrWindow::]SetCurrentZoom , *ZoomFactor* [, /RESET]

IDLgrWindow::SetProperty - Sets the value of a property or group of properties for the window.

Obj->[IDLgrWindow::]SetProperty [, *PROPERTY*=*value*]

IDLgrWindow::Show - Exposes or hides a window.

Obj->[IDLgrWindow::]Show, *Position*

IDLgrWindow::ZoomIn - causes the current zoom factor for this window to be increased (that is, multiplied by the factor given by the window's ZOOM_BASE property).

Obj->[IDLgrWindow::]ZoomIn

IDLgrWindow::ZoomOut - causes the current zoom factor for this window to be decreased (that is, divided by the factor given by the window's ZOOM_BASE property).

Obj->[IDLgrWindow::]ZoomOut

IDLitCommand - The base functionality for the iTools command buffer system.

Properties: [, SKIP_REDO{Get, Init, Set}=0 | 1] [, SKIP_UNDO{Get, Init, Set}=0 | 1] [, TARGET_IDENTIFIER{Get, Init, Set}=*string*] [, OPERATION_IDENTIFIER{Get, Init, Set}=*string*]

IDLitCommand::AddItem - Adds the specified data item to the data dictionary associated with this object.

Result = Obj->[IDLitCommand::]AddItem(*StrItem*, *Item* [, /OVERWRITE])

IDLitCommand::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or

Obj-> [IDLitCommand::]Cleanup()

IDLitCommand::GetItem - Retrieves the specified item from the data dictionary associated with this object.

Result = *Obj*->[IDLitCommand::]GetItem(*StrItem*, *Item*)

IDLitCommand::GetProperty - Retrieves the value of a property or group of properties of a command object.

Obj->[IDLitCommand::]GetProperty [, PROPERTY=*variable*]

IDLitCommand::GetSize - Returns an approximate value for the amount of memory being used by the items in the data dictionary associated with this object.

Result = *Obj*->[IDLitCommand::]GetSize([, /KILOBYTES])

IDLitCommand::Init - Initializes the object and allows specification of items associated with it.

Obj = OBJ_NEW('IDLitCommand' [, *PROPERTY*=*value*])

or

Result = *Obj*->[IDLitCommand::]Init([*PROPERTY*=*value*])

IDLitCommand::SetProperty - Sets the value of a property or group of properties for the command object.

Obj->[IDLitCommand::]SetProperty [, *PROPERTY*=*value*]

IDLitCommandSet - A container for IDLitCommand objects, which allows a group of commands to be managed as a single item.

IDLitCommandSet::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or

Obj-> [IDLitCommandSet::]Cleanup()

IDLitCommandSet::GetSize - Returns an approximate value for the amount of memory being used by the items contained by this command set.

Result = *Obj*->[IDLitCommand::]GetSize([/KILOBYTES])

IDLitCommandSet::Init - Initializes the object.

Obj = OBJ_NEW('IDLitCommandSet')

or

Result = *Obj*->[IDLitCommandSet::]Init()

IDLitComponent - A core or base component, from which all other components subclass.

Properties: [, COMPONENT_VERSION{Get}=value] [, DESCRIPTION{Get, Init, Set}=string] [, ICON{Get, Init, Set}=string] [, HELP{Get, Init, Set}=string] [, IDENTIFIER{Get, Init, Set}=string] [, NAME{Get, Init, Set}=string] [, /PRIVATE{Get, Init, Set}] [, UVALUE{Get, Init, Set}=value]

IDLitComponent::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or

Obj->[IDLitComponent::]Cleanup

IDLitCompoment::EditUserDefProperty - Defines the interface that is displayed when a user selects the “Edit” button on a user-defined property in the property sheet interface.

Result = *Obj*->[IDLitComponent::]EditUserDefProperty(*iTool*, *PropertyIdentifier*)

IDLitComponent::GetFullIdentifier - Navigates the iTool object container hierarchy of the object on which it is called and retrieves the fully-qualified object identifier.

Result = *Obj*->[IDLitComponent::]GetFullIdentifier([*Objref*])

IDLitComponent::GetProperty - Retrieves the value of an IDLitComponent property or properties.

Obj->[IDLitComponent::]GetProperty [, *PROPERTY*=*variable*]

IDLitComponent::GetPropertyAttribute - Retrieves property attribute values for a registered property.

Obj->[IDLitComponent::]GetPropertyAttribute, *PropertyIdentifier* [, TYPE=*variable*]

IDLitComponent::GetPropertyByIdentifier - Retrieves the value of an IDLitComponent property.

Result = *Obj*->[IDLitComponent::]GetPropertyByIdentifier(*PropertyIdentifier*, *Value*)

IDLitComponent::Init - Initializes the IDLitComponent object

Obj = OBJ_NEW('IDLitComponent' [, *PROPERTY*=*value*])

or

Result = *Obj*->[IDLitComponent::]Init([, *PROPERTY*=*value*])

IDLitComponent::QueryProperty - Checks whether a property identifier is registered, or retrieves a list of all registered properties.

Result = *Obj*->[IDLitComponent::]QueryProperty([*PropertyIdentifier*])

IDLitComponent::RegisterProperty - Registers a property as belonging to the component.

Obj->[IDLitComponent::]RegisterProperty, *PropertyIdentifier* [, *Type*] [, /BOOLEAN] [, /COLOR] [, DESCRIPTION=*string*] [, ENUMLIST=*stringvector*] [, /FLOAT] [, /HIDE] [, /INTEGER] [, /LINESTYLE] [, NAME=*string*] [, /SENSITIVE] [, /STRING] [, /SYMBOL] [, /THICKNESS] [, /UNDEFINED] [, USERDEF=*string*] [, VALID_RANGE=*vector*]

IDLitComponent::Restore - Performs any transitional work required after an object has been restored from a SAVE file.

Obj->[IDLitComponent::]Restore

IDLitComponent::SetProperty - Sets the value of an IDLitComponent property or properties.

Obj->[IDLitComponent::]SetProperty [, *PROPERTY*=*value*]

IDLitComponent::SetPropertyAttribute - Sets one or more property attributes for a registered property.

Obj->[IDLitComponent::]SetPropertyAttribute, *PropertyIdentifier*

IDLitComponent::SetPropertyByIdentifier - Sets the value of an IDLitComponent property.

Obj->[IDLitComponent::]SetPropertyByIdentifier, *PropertyIdentifier*, *Value*

IDLitComponent::UpdateComponentVersion - Updates the value of the COMPONENT_VERSION property for the specified object to the current release of IDL.

Obj->[IDLitComponent::]UpdateComponentVersion

IDLitContainer - A specialization of the IDL_Container class that manages a collection of IDLitComponents and provides methods for working with the Identifier system of the iTools framework.

IDLitContainer::Add - Adds items to the container object.

Obj->[IDLitContainer::]Add, *Components* [, /NO_NOTIFY]

IDLitContainer::AddByIdentifier - Adds an object to the container hierarchy in the position specified by the *Identifier* argument.

Obj->[IDLitContainer::]AddByIdentifier, *Identifier*, *Item* [, FOLDER_CLASSNAME=*string*]

IDLitContainer::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLitContainer::]Cleanup

IDLitContainer::FindIdentifiers - Retrieves the full identifiers for items within an iTool container.

Result = *Obj*->IDLitContainer::FindIdentifiers([*Pattern*] [, COUNT=*variable*] [, /LEAF_NODES])

IDLitContainer::Get - Retrieves items from the container.

Result = *Obj*->[IDLitContainer::]Get([, /ALL] [, COUNT=*variable*] [, ISA=*string* or *array of strings*] [, POSITION=*index* or *array of indices*] [, /SKIP_PRIVATE])

IDLitContainer::GetByIdentifier - Retrieves an object from a container hierarchy using the specified identifier to locate the object.

Result = *Obj*->[IDLitContainer::]GetByIdentifier(*Identifier*)

IDLitContainer::Init - Initializes the object.

Obj = OBJ_NEW('IDLitContainer') or *Result* = *Obj*->[IDLitContainer::]Init()

IDLitContainer::Remove - Removes items from the container.

Obj->[IDLitContainer::]Remove, *Components* [, /NO_NOTIFY]

IDLitContainer::RemoveByIdentifier - Removes an object from a container hierarchy using the specified identifier to locate the object.

Result = *Obj*->[IDLitContainer::]RemoveByIdentifier(*Identifier*)

IDLitData - A generic data storage object that can hold any IDL data type available. It provides typing, metadata, and data change notification functionality. When coupled with IDLitDataContainer, it forms the element for the construction of composite data types.

Properties: [, /HIDE{Get, Init, Set}] [, READ_ONLY{Init}] [, TYPE{Get, Init}=*string*]

IDLitData::AddDataObserver - Specifies an object (the *Observer*) that will be notified when the contents of the data object are changed.

Obj->[IDLitData::]AddDataObserver, *Observer*

IDLitData::Cleanup - Performs all cleanup operations on the object

OBJ_DESTROY, *Obj* or *Obj*->[IDLitData::]Cleanup

IDLitData::Copy - Returns an exact copy of the data object and its contents, including registered property values.

Result = *Obj*->[IDLitData::]Copy()

IDLitData::GetByType - Returns all contained objects of the specified iTool data type.

Result = *Obj*->[IDLitData::]GetByType(*Type* [, COUNT=*variable*])

IDLitData::GetData - Retrieves the data stored in the object.

Result = *Obj*->[IDLitData::]GetData(*Data*[, *Identifier*] [, NAN=*variable*] [, /NO_COPY])

IDLitData::GetProperty - Retrieves the value of an IDLitData property.

Obj->[IDLitData::]GetProperty[, *PROPERTY=variable*]

IDLitData::GetSize - Returns an approximate value for the amount of memory being used by the data object.

Result = *Obj*->[IDLitData::]GetSize()

IDLitData::Init - Initializes the IDLitData object.

Obj = OBJ_NEW('IDLitData'[, *Data*] [, *PROPERTY=value*]) or

Result = *Obj*->[IDLitData::]Init([*Data*] [, *PROPERTY=value*])

IDLitData::NotifyDataChange - Is called when a data object has been changed; it is part of the notification process that allows data updates to be reflected by visualizations that use the data. It works in conjunction with the NotifyDataComplete method to provide a two-pass change notification system that minimizes the number of operations performed when a data object changes.

Obj->[IDLitData::]NotifyDataChange

IDLitData::NotifyDataComplete - Is called after a data object has been changed; it is part of the notification process that allows data updates to be reflected by visualizations that use the data. It works in conjunction with the NotifyDataChange method to provide a two-pass change notification system that minimizes the number of operations performed when a data object changes.

Obj->[IDLitData::]NotifyDataComplete

IDLitData::RemoveDataObserver - Unregisters an object that has been registered as an *observer* of this data object.

Obj->[IDLitData::]RemoveDataObserver, *Observer*

IDLitData::SetData - Copies the data from an IDL variable or expression into the data object, and notifies all its observers that the data has changed.

Result = *Obj*->[IDLitData::]SetData(*Data*[, *Identifier*] [, /NO_COPY] [, /NULL])

IDLitData::SetProperty - Sets the value of an IDLitData property

Obj->[IDLitData::]SetProperty[, *PROPERTY=value*]

IDLitDataContainer - A container for IDLitData and IDLitDataContainer objects. This container is used to form hierarchical data structures. Data and DataContainer objects can be added and removed to/from a DataContainer during program execution, allowing for dynamic creation of composite data types.

IDLitDataContainer::Add - Adds items to the data container object.

Obj->[IDLitDataContainer::]Add, Data [, /NO_NOTIFY]

IDLitDataContainer::Cleanup - Performs all cleanup operations on the object.

OBJ_DESTROY, *Obj* or

Obj->[IDLitDataContainer::]Cleanup

IDLitDataContainer::GetData - Retrieves the data value contained in the data object specified by the *Identifier* argument.

Result = *Obj*->[IDLitDataContainer::]GetData(*Data* [, *Identifier*] [, /NO_COPY])

IDLitDataContainer::GetIdentifiers - Retrieves the object identifiers for all data and data container objects contained in the data container object.

Result = *Obj*->[IDLitDataContainer::]GetDataIdentifiers([*Pattern*] [, /LEAF])

IDLitDataContainer::GetProperty - Retrieves the value of an IDLitDataContainer property

Obj->[IDLitDataContainer::]GetProperty [, *PROPERTY=variable*]

IDLitDataContainer::Init - Initializes the IDLitDataContainer object.

Obj = OBJ_NEW('IDLitDataContainer'[, *Data*] [, *PROPERTY=value*]) or

Result = *Obj*->[IDLitDataContainer::]Init([*Data*] [, *PROPERTY=value*])

IDLitDataContainer::SetData - Stores data in the IDLitData object specified by *Identifier.*

Result = *Obj*->[IDLitDataContainer::]SetData(*Data*, *Identifier*[, /NO_COPY] [, /NULL])

IDLitDataContainer::SetProperty - Sets the value of an IDLitDataContainer property.

Obj->[IDLitDataContainer::]SetProperty [, *PROPERTY=value*]

IDLitDataOperation - A subclass to IDLitOperation that automates data access and automatically records information for the undo-redo system.

Properties: [, WITHIN_UI{GET}=1 | 0]

IDLitDataOperation::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or

Obj->[IDLitOperation::]Cleanup

IDLitDataOperation::DoExecuteUI - Provides a way for the iTool developer to request user input before performing an operation.

Result = *Obj*->[IDLitDataOperation::]DoExecuteUI()

IDLitDataOperation::Execute - Contains the execution logic for the operation.

Result = *Obj*->[IDLitDataOperation::]Execute(*Data*)

IDLitDataOperation::GetProperty - Retrieves the value of a property or group of properties of an operation object.

Obj->[IDLitDataOperation::]GetProperty [, *PROPERTY=variable*]

IDLitDataOperation::Init - Initializes the IDLitDataOperation object and sets properties that define the behavior the operation provides.

Obj = OBJ_NEW('IDLitDataOperation' [, *PROPERTY=variable*]) or

Result = *Obj*->[IDLitDataOperation::]Init([, *PROPERTY=variable*])

IDLitDataOperation::SetProperty - Sets the value of a property or group of properties for the operation.

Obj->[IDLitDataOperation::]SetProperty [, *PROPERTY=value*]

IDLitDataOperation::UndoExecute - IS called when a user selects the Undo operation after executing an IDLitDataOperation that sets the value of the REVERSIBLE_OPERATION property to 1.

Result = *Obj*->[IDLitDataOperation::]UndoExecute(*Data*)

IDLitIMessaging - An interface providing common methods to send or trigger messaging and error actions, which may occur during execution.

IDLitIMessaging::AddOnNofityObserver - Is used to register a specified iTool component object to receive messages generated by the DoOnNotify method of another specified iTool component object.

Obj->[IDLitIMessaging::]AddOnNotifyObserver, *IdObserver*, *IdSubject*

IDLitIMessaging::DoOnNotify - Is used to broadcast a notification message to iTool component objects that are observing the source of the message.

Obj->[IDLitIMessaging::]DoOnNotify, *IdOriginator*, *IdMessage*, *Value*

IDLitIMessaging::ErrorMessage - Is used to display an error message to the user.

Obj->[IDLitIMessaging::]ErrorMessage, *StrMessage* [, SEVERITY=*integer*] [, TITLE=*string*] [, /USE_LAST_ERROR]

IDLitIMessaging::GetTool - Returns an object reference to the iTool object associated with the object on which it is called.

Result = *Obj*->[IDLitIMessaging::]GetTool()

IDLitIMessaging::ProbeStatusMessage - Is used to display a status message to the user, which is displayed in a data-specific region of the user interface.

Obj->[IDLitIMessaging::]ProbeStatusMessage, *StrMessage*

IDLitIMessaging::ProgressBar - Is used to display a progress bar to the user and update the displayed values.

Result = *Obj*->[IDLitIMessaging::]ProgressBar(*StrMessage* [, CANCEL=*string*] [, PERCENT=*value*] [, /SHUTDOWN])

IDLitIMessaging::PromptUserText - Is used to prompt the iTool user with a question and retrieve a text answer.

Result = *Obj*->[IDLitIMessaging::PromptUserText(*StrPrompt*, *Answer* [, TITLE=*string*])

IDLitIMessaging::PromptUserYesNo - Is used to prompt the user with a yes or no question and return an answer.

Result = *Obj*->[IDLitIMessaging::PromptUserYesNo(*StrPrompt*, *Answer* [, TITLE=*string*])

IDLitIMessaging::RemoveOnNotifyObserver - Is used to un-register a specified iTool component object as wishing to receive messages generated by the DoOnNotify method of another specified iTool component object.

Obj->[IDLitIMessaging::]RemoveOnNotifyObserver, *IdObserver*, *IdSubject*

IDLitIMessaging::SignalError - Is used to signal an error in the system.

Obj->[IDLitIMessaging::]SignalError, *StrMessage* [, CODE=*string*] [, SEVERITY=*integer*]

IDLitIMessaging::StatusMessage - Is used to display a status message to the user.

Obj->[IDLitIMessaging::]StatusMessage, *StrMessage* [, SEGMENT_IDENTIFIER=*string*]

IDLitManipulator - The base functionality of the iTools manipulator system.

Properties: [, /BUTTON_EVENTS{Get, Init, Set}] [, DEFAULT_CURSOR{Init}=*string*] [, DESCRIPTION{Get, Init, Set}=*string*] [, /DISABLE{Get, Init, Set}] [, DRAG_QUALITY{Get, Init, Set}={0 | 1 | 2}] [, /KEYBOARD_EVENTS{Get, Init, Set}] [, /MOTION_EVENTS{Get, Init, Set}] [, OPERATION_IDENTIFIER{Get, Init, Set}=*string*] [, PARAMETER_IDENTIFIER{Get, Init, Set}=*string*}] [, TRANSIENT_DEFAULT{Get, Init, Set}={0 | 1 | 2 | 3}] [, /TRANSIENT_MOTION{Get, Init, Set}] [, TYPES{Get, Init}=*string vector*] [, /VIEWS_ONLY{Init}] [VISUAL_TYPE{Get, Init, Set}=*string*]

IDLitManipulator::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLitManipulator::]Cleanup()

IDLitManipulator::CommitUndoValues - Is used to complete a transaction that is occurring as a result of the manipulator interaction.

Result = *Obj*->[IDLitManipulator::]CommitUndoValues([/UNCOMMENT])

IDLitManipulator::GetCursorType - Retrieves the name of the cursor to display for the manipulator.

Result = *Obj*->[IDLitManipulator::]GetCursorType(*TypeIn*, *KeyMods*)

IDLitManipulator::GetProperty - Retrieves the value of an IDLitManipulator property

Obj->[IDLitManipulator::]GetProperty [, *PROPERTY*=*variable*]

IDLitManipulator::Init - Initializes the IDLitManipulator object

Obj = OBJ_NEW('IDLitManipulator' [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLitManipulator::]Init([, *PROPERTY*=*value*])

IDLitManipulator::OnKeyboard - Is used when a keyboard event occurs on the target window (*Win*).

Obj->[IDLitManipulator::]OnKeyboard, *Win*, *IsASCII*, *Character*, *Key Value*, *X*, *Y*, *Press*, *Release*, *KeyMods*

IDLitManipulator::OnLoseCurrentManipulator - Is used when this manipulator is no longer the current manipulator in the system.

Obj->[IDLitManipulator::]OnLoseCurrentManipulator

IDLitManipulator::OnMouseDown - Is used when a mouse down event occurs on the target window.

Obj->[IDLitManipulator::]OnMouseDown, *Win*, *X*, *Y*, *IButton*, *KeyMods*, *NClicks*

IDLitManipulator::OnMouseMotion - Manages the setting of the cursor on the window if no mouse button is down.

Obj->[IDLitManipulator::]OnMouseMotion, *Win*, *X*, *Y*, *KeyMods*

IDLitManipulator::OnMouseUp - Is used when a mouse up event occurs on the target window.

Obj->[IDLitManipulator::]OnMouseUp, *Win*, *X*, *Y*, *IButton*

IDLitManipulator::RecordUndoValues - Is used to begin recording the transaction that is occurring as a result of the manipulator interaction. This method works in conjunction with the CommitUndoValues method.

Result = *Obj*->[IDLitManipulator::]RecordUndoValues()

IDLitManipulator::RegisterCursor - Defines the appearance and name of the cursor associated with the manipulator.

Obj->[IDLitManipulator::]RegisterCursor, *ArrCursor*, *Name* [, /DEFAULT]

IDLitManipulator::SetCurrentManipulator - Sets the manipulator as the current manipulator in the system.

Obj->[IDLitManipulator::]SetCurrentManipulator

IDLitManipulator::SetProperty - Sets the value of an IDLitManipulator property

Obj->[IDLitManipulator::]SetProperty [, *PROPERTY=value*]

IDLitManipulatorContainer - A container for IDLitManipulator objects, which allows for the construction of manipulator hierarchies. This container implements the concept of a current manipulator for the items it contains.

Property: [, /AUTO_SWITCH{Get, Init, Set}]

IDLitManipulatorContainer::Add - Is used to add a new manipulator to the container.

Obj->[IDLitManipulatorContainer::]Add, *Manipulator*

IDLitManipulatorContainer::GetCurrent - Is used to get the object reference of the current manipulator of the container.

Result = *Obj*->[IDLitManipulatorContainer::] GetCurrent()

IDLitManipulatorContainer::GetCurrentManipulator - Is used to get the current manipulator of the system.

Result = *Obj*->[IDLitManipulatorContainer::] GetCurrentManipulator([, /IDENTIFIER])

IDLitManipulatorContainer::GetProperty - Retrieves the value of an IDLitManipulatorContainer property

Obj->[IDLitManipulatorContainer::]GetProperty [, *PROPERTY=variable*]

IDLitManipulatorContainer::Init - Initializes the IDLitManipulatorContainer object.

Obj = OBJ_NEW('IDLitManipulatorContainer' [, *PROPERTY=value*]) or

Result = *Obj*->[IDLitManipulatorContainer::]Init([*PROPERTY=value*])

IDLitManipulatorContainer::OnKeyboard - Is used when a keyboard event occurs on the target window (*Win*).

Obj->[IDLitManipulator::]OnKeyboard, *Win*, *IsASCII*, *Character*, *KeyValue*, *X*, Y, *Press*, *Release*, *KeyMods*

IDLitManipulatorContainer::OnMouseDown - Is used when a mouse down event occurs on the target window.

Obj->[IDLitManipulatorContainer::]OnMouseDown, *Win*, *X*, *Y*, *IButton*, *KeyMods*, *NClicks*

IDLitManipulatorContainer::OnMouseMotion - Manages the setting of the cursor on the window if no mouse button is down.

Obj->[IDLitManipulatorContainer::]OnMouseMotion, *Win*, *X*, *Y*, *KeyMods*

IDLitManipulatorContainer::OnMouseUp - Is used when a mouse up event occurs on the target window.

Obj->[IDLitManipulatorContainer::]OnMouseUp, *Win*, *X*, *Y*, *IButton*

IDLitManipulatorContainer::SetCurrent - Is used to set a manipulator within the container to be the current manipulator.

Obj->[IDLitManipulatorContainer::]SetCurrent, *Manipulator*

IDLitManipulatorContainer::SetCurrentManipulator - Is used to set the current child manipulator in a manipulator hierarchy.

Obj->[IDLitManipulatorContainer::] SetCurrentManipulator, *Identifier*

IDLitManipulatorContainer::SetProperty - Sets the value of an IDLitManipulatorContainer property

Obj->[IDLitManipulator::]SetProperty [, *PROPERTY=value*]

IDLitManipulatorManager - A specialization of the manipulator container (IDLitManipulatorContainer), which acts as the root of the manipulator hierarchy.

IDLitManipulatorManager::Add - Is used to add manipulators to the manipulator manager.

Obj->[IDLitManipulatorManager::]Add, *Manipulator* [, /DEFAULT]

IDLitManipulatorManager::AddManipulatorObserver - Is used to add an observer to the manipulator system.

Obj->[IDLitManipulatorManager::] AddManipulatorObserver, *Observer*

IDLitManipulatorManager::GetDefaultManipulator - Returns a reference to the manipulator that was most recently added as the default manipulator.

Result = *Obj*->[IDLitManipulatorManager::]-> GetDefaultManipulator()

IDLitManipulatorManager::Init - Initializes the IDLitManipulatorManager object

Obj = OBJ_NEW('IDLitManipulatorManager' [, *PROPERTY=value*]) or

Result = *Obj*->[IDLitManipulatorManager::]Init([*PROPERTY=value*])

IDLitManipulatorManager::RemoveManipulatorObserver - Is used to remove a manipulator observer from the manipulator object.

Obj->[IDLitManipulatorManager::] RemoveManipulatorObserver, *Observer*

IDLitManipulatorVisual - The basis of all visual elements associated with an interactive manipulator.

Properties: [, /UNIFORM_SCALE] [, VISUAL_TYPE=*string*]

IDLitManipulatorVisual::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or

Obj->[IDLitManipulatorVisual::]Cleanup

IDLitManipulatorVisual::GetProperty - Retrieves the value of an IDLitManipulatorVisual property.

Obj->[IDLitManipulatorVisual::]GetProperty

IDLitManipulatorVisual::Init - Initializes the IDLitManipulatorVisual object.

Obj = OBJ_NEW('IDLitManipulatorVisual' [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLitManipulatorVisual::]Init([, *PROPERTY*=*value*])

IDLitManipulatorVisual::SetProperty - Sets the value of an IDLitManipulatorVisual property.

Obj->[IDLitManipulatorVisual::]SetProperty [, *PROPERTY*=*value*]

IDLitOperation - The basis for all iTool operations. It defines how an operation is executed and how information about the operation is recorded for the command transaction (undo-redo) system.

Properties: [, /EXPENSIVE_COMPUTATION{Get, Init}] [, /REVERSIBLE_OPERATION{Get, Init}] [, /SHOW_EXECUTION_UI{Get, Init}] [, TYPES{Get, Init}=*string or string array*]

IDLitOperation::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or

Obj->[IDLitOperation::]Cleanup

IDLitOperation::DoAction - Is called when an operation is requested by the iTools system, either as the result of a user action such as selection of a menu item or toolbar button, or by another operation.

Result = *Obj*->[IDLitOperation::]DoAction(*Tool*)

IDLitOperation::GetProperty - Retrieves the value of a property or group of properties of an operation object.

Obj->[IDLitOperation::]GetProperty [, *PROPERTY*=*variable*]

IDLitOperation::Init - Initializes the IDLitOperation object and sets properties that define the behavior the operation provides.

Obj = OBJ_NEW('IDLitOperation' [, *PROPERTY*=*variable*]) or

Result = *Obj*->[IDLitOperation::]Init([, *PROPERTY*=*variable*])

IDLitOperation::QueryAvailability - Determines whether an operation is applicable for the selected data and/or visualization.

Result = *Obj*->[IDLitOperation::]QueryAvailability(*Tools*, *SelTypes*)

DLitOperation::RecordFinalValues - Records the information needed to redo an operation.

Result = *Obj*->[IDLitOperation::]RecordFinalValues(*CommandSet*, *Targets* [, *IdProperty*])

IDLitOperation::RecordInitialValues - Records the information needed to undo an operation.

Result = *Obj*->[IDLitOperation::]RecordInitialValues(*CommandSet*, *Targets*[, *IdProperty*])

IDLitOperation::RedoOperation - Is called by the iTool system when the user requests the re-execution of an operation (usually by selecting Redo from the iTool Edit menu or toolbar).

Result = *Obj*->[IDLitOperation::]RedoOperation(*CommandSet*)

IDLitOperation::SetProperty - Sets the value of a property or group of properties for the operation.

Obj->[IDLitOperation::]SetProperty [, *PROPERTY*=*value*]

IDLitOperation::UndoOperation - Is called by the iTool system when the user requests the un-execution of an operation (usually by selecting Undo from the iTool Edit menu or toolbar).

Result = *Obj*->[IDLitOperation::]UndoOperation(*CommandSet*)

IDLitParameter - An interface providing parameter management methods to associate parameter names with IDLitData objects.

IDLitParameter::Cleanup - Removes the *data observer* from each data object in the visualization object's parameter set, and cleans up the objects and pointers defined to hold parameter data when the visualization object was created.

OBJ_DESTROY, *Obj* or

Obj->[IDLitParameter::]Cleanup()

IDLitParameter::GetParameter - Retrieves the IDLitData object associated with a registered parameter.

Result = *Obj*->[IDLitParameter::]GetParameter(*Name* [, /ALL] [, COUNT=*variable*])

IDLitParameter::GetParameterAttribute - Retrieves the value of an attribute associated with a registered parameter.

Obj->[IDLitParameter::]GetParameterAttribute, *ParamName* [, ATTRIBUTE=*variable*]

IDLitParameter::GetParameterSet - Returns a reference to the IDLitParameterSet object associated with the visualization object.

Result = *Obj*->[IDLitParameter::]GetParameterSet([, /DEEP_COPY)

IDLitParameter::Init - Initializes object instance fields that contain parameter data.

Result = *Obj*->[IDLitParameter::]Init()

IDLitParameter::OnDataChangeUpdate - Is called when a data value has been updated or a new data object has been associated with the visualization object.

Obj->[IDLitParameter::]OnDataChangeUpdate, *Data*, *ParameterName*

IDLitParameter::OnDataDisconnect - Is called when a data value has been disconnected from a parameter.

Obj->[IDLitParameter::]OnDataDisconnect, *ParameterName*

IDLitParameter::QueryParamter - Checks whether a parameter is registered, or retrieves a list of the of all registered parameters.

Result = *Obj*->[IDLitParameter::]QueryParameter(*ParamName* [, COUNT=*variable*]

IDLitParameter::RegisterParameter - Registers a parameter with the visualization object.

Obj->[IDLitParameter::]RegisterParameter, *Name* [, /BY_VALUE] [, DESCRIPTION=*string*] [, /INPUT] [, /OPTARGET] [, /OPTIONAL] [, /OUTPUT] [, TYPES=*string*]

IDLitParameter::SetData - Is used to set data in this interface, associating a data object with a given parameter.

Result = *Obj*->[IDLitParameter::]SetData(*Data* [, /BY_VALUE] [, PARAMETER_NAME=*string*] [, /NO_UPDATE]

IDLitParameter::SetParameterAttribute - Sets one or more parameter attributes for a registered parameter.

Obj->[IDLitParameter::]SetParameterAttribute, *ParamName* [, ATTRIBUTE=*variable*]

IDLitParameter::SetParameterSet - Is used to associate an IDLitParameterSet object with the visualization object's parameter interface.

Result = *Obj*->[IDLitParameter::]SetParameterSet(*ParamSet*)

IDLitParameterSet - A specialized subclass of the IDLitDataContainer class. This class provides the ability to associate names with contained IDLitData objects.

IDLitParameterSet::Add - Is used to add data to the parameter set.

Obj->[IDLitParameterSet::]Add, *Data* [, PARAMETER_NAME=*string*] [, /PRESERVE_LOCATION]

IDLitParameterSet::Cleanup - Performs all cleanup on the parameter set object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLitParameterSet::]Cleanup()

IDLitParameterSet::Copy - Returns a copy of the parameter set and its contents.

Result = *Obj*->[IDLitParameterSet::]Copy()

IDLitParameterSet::Get - Is used to retrieve one or more IDLitData objects from the parameter set.

Result = *Obj*->[IDLitParameterSet::]Get([, /ALL] [, COUNT=*variable*] [, NAME=*variable*] [, POSITION=*integer*])

IDLitParameterSet::GetByName - Returns the IDLitData object associated with a specified named parameter.

Result = *Obj*->[IDLitParameterSet::]GetByName(*Names* [, COUNT=*variable*] [, NAME=*variable*])

IDLitParameterSet::GetParameterName - Retrieves the name of a specified parameter using a provided data object.

Result = *Obj*->[IDLitParameterSet::] GetParameterName(*Data*, *Name*)

IDLitParameterSet:Init: - Initializes the IDLitParameter object.

Obj = OBJ_NEW('IDLitParameterSet') or *Result* = *Obj*->[IDLitParameterSet::]Init()

IDLitParameterSet::Remove - Is used to remove a data item from the parameter set.

Obj->[IDLitParameterSet::]Remove [, *Items*] [, /ALL] [, POSITION=*integer*]

IDLitReader - The definition of the interface and the process used to construct file readers for the iTools framework. When a new file reader is constructed for the iTools system, a new class is subclassed from this IDLitReader class.

IDLitReader::Cleanup - Performs all cleanup on the object, and should be called by the subclass' Cleanup method.

OBJ_DESTROY, *Obj* or *Obj*->[IDLitReader::]Cleanup

IDLitReader::GetData - Is called by the system to retrieve the data from the current file.

Result = *Obj*->[IDLitReader::]GetData(*Data*)

IDLitReader::GetFileExtensions - Is called by the system to retrieve the file extensions supported by this particular reader.

Result = *Obj*->[IDLitReader::]GetFileExtensions([COUNT=*variable*])

IDLitReader::GetFilename - Is called by the system to retrieve the current filename associated with this reader.

Result = *Obj*->[IDLitReader::]GetFilename()

IDLitReader::GetProperty - Retrieves the value of an IDLitReader property.

Obj->[IDLitReader::]GetProperty [, *PROPERTY*=*variable*]

IDLitReader::Init - Initializes the IDLitReader object.

Obj = OBJ_NEW('IDLitReader', [, *Extensions*] [, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLitReader::]Init([, *Extensions*] [, *PROPERTY*=*value*])

IDLitReader::IsA - Is called by the system to determine if the given file is of the type supported by this file reader.

Result = *Obj*->[IDLitReader::]IsA(*Filename*)

IDLitReader::SetFilename - Is called by the system to set the current filename associated with this reader.

Obj->[IDLitReader::]SetFilename, *Filename*

IDLitReader::SetProperty - Sets the value of an IDLitReader property.

Obj->[IDLitReader::]SetProperty[, *PROPERTY*=*value*]

IDLitTool - All the functionality provided by a particular instance of an IDL Intelligent Tool (iTool). This object provides the management systems for the underlying tool functionality.

Properties: [, DESCRIPTION{Get, Init, Set}=*string*] [, ICON{Get, Init, Set}=*string*] [, NAME{Get, Init, Set}=*string*] [TYPE{Init, Set}=*string or string array*] [, /UPDATE_BYTYPE{Init, Set}] [, /VERBOSE{Get, Init, Set}] [, VERSION{Get, Init}=*variable*]

IDLitTool::ActivateManipulator - Activates a manipulator that has been registered with this tool.

Obj->ActivateManipulator, Identifier[, /DEFAULT]

IDLitTool::Add - Adds any item to the tool.

Obj->[IDLitTool::]Add, *Item*

IDLitTool::AddService - Adds a service to the tool.

Obj->[IDLitTool::]AddService, *Service*

IDLitTool::Cleanup - Performs all cleanup on the object

OBJ_DESTROY, *Obj* or *Obj*->[IDLitTool::]Cleanup

IDLitTool::CommitActions - Commits all pending transactions to the undo-redo buffer and causes a refresh of the current window.

Obj->[IDLitTool::]CommitActions

IDLitTool::DisableUpdates - Disables all drawing updates to the current window and UI updates (menu sensitivity) being passed to the user interface.

Obj->[IDLitTool::]DisableUpdates ([, PREVIOUSLY_DISABLED=*variable*])

IDLitTool::DoAction - Initiates an operation or action in the tool object.

Result = *Obj*->[IDLitTool::]DoAction(*Identifier*)

IDLitTool::DoSetProperty - Sets a property on a target component object, and places the change in the undo/redo transaction buffer.

Result = *Obj*->[IDLitTool::]DoSetProperty(*TargetIdentifier*, *PropertyIdentifier*, *Value*)

IDLitTool::DoUIService - Initiates a request for a UI service to execute.

Result = *Obj*->[IDLitTool::]DoUIService(*ServiceIdentifier*, *Requestor*)

IDLitTool::EnableUpdates - Re-enables all drawing updates to the current window and UI updates (menu sensitivity) being passed to the user interface.

Obj->[IDLitTool]::EnableUpdates

IDLitTool::FindIdentifiers - Retrieve the full identifiers for items within the tool container

Result = *Obj*->IDLitTool::FindIdentifiers([*Pattern*] [, /ANNOTATIONS] [, /COUNT=variable] [, /DATA_MANAGER] [, /FILE_READERS] [, /FILE_WRITERS] [, /LEAF_NODES] [, /MANIPULATORS] [, /OPERATIONS] [, /VISUALIZATIONS])

IDLitTool::GetCurrentManipulator - Returns the current manipulator in the system.

Result = *Obj*->[IDLitTool]::GetCurrentManipulator()

IDLitTool::GetFileReader - Retrieves a file reader registered with the tool object.

Result = *Obj*->[IDLitTool::]GetFileReaders(*Identifier* [, /ALL] [, COUNT=*variable*])

IDLitTool::GetFileWriter - Retrieves a file writer registered with the tool object.

Result = *Obj*->[IDLitTool::]GetFileWriters(*Identifier* [, /ALL] [, COUNT=*variable*])

IDLitTool::GetManipulators - Retrieves the manipulators registered with the tool object.

Result = *Obj*->[IDLitTool::]GetManipulators([COUNT=*variable*])

IDLitTool::GetOperations - Retrieves the operations registered with the tool object.

Result = *Obj*->[IDLitTool::]GetOperations([, IDENTIFIER=*string*] [, COUNT=*variable*])

IDLitTool::GetProperty - Retrieves the value of an IDLitTool property.

Obj->[IDLitTool::]GetProperty, [*PROPERTY*=*variable*]

IDLitTool::GetSelectedItems - Returns a vector of references to the objects currently selected within the current window in the tool.

Result = *Obj*->[IDLitTool::]GetSelectedItems([COUNT=*variable*])

IDLitTool::GetService - Retrieves a service that has been registered with the tool.

Result = *Obj*->[IDLitTool::]GetService(*IdService*)

IDLitTool::GetVisualization - Retrieves a visualization registered with the tool object.

Result = *Obj*->[IDLitTool::]GetVisualizations(*Identifier* [, /ALL] [, COUNT=*variable*])

IDLitTool::Init - Initializes the IDLitTool object

Obj = OBJ_NEW('IDLitTool'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLitTool::]Init([*PROPERTY*=*value*])

IDLitTool::RefreshCurrentWindow - Redraws the current window of the tool.

Obj->[IDLitTool]::RefreshCurrentWindow

IDLitTool::Register - Registers a generic component with the tool.

Obj->[IDLitTool::]Register, *Name*, *ClassName* [, /DEFAULT] [, DESCRIPTION=*string*] [, ICON=*string*] [, IDENTIFIER=*string*] [, PROXY=*string*]

IDLitTool::RegisterCustomization - Registers an operation class that represents the graphics customization operation to be associated with this tool.

Obj->[IDLitTool::]RegisterCustomization, Name, ClassName

IDLitTool::RegisterFileReader - Registers a file reader component with the tool.

Obj->[IDLitTool::]RegisterFileReader, *Name*, *ClassName* [, /DEFAULT][, DESCRIPTION=*string*]

[, ICON=*string*] [, IDENTIFIER=*string*]
[, PROXY=*string*]

IDLitTool::RegisterFileWriter - Registers a file writer component with the tool.

Obj->[IDLitTool::]RegisterFileWriter, *Name*, *ClassName*
[, /DEFAULT][, DESCRIPTION=*string*]
[, ICON=*string*] [, IDENTIFIER=*string*]
[, PROXY=*string*]

IDLitTool::RegisterManipulator - Registers a manipulator component with the tool.

Obj->[IDLitTool::]Manipulator, *Name*, *ClassName*
[, /DEFAULT] [, DESCRIPTION=*string*]
[, ICON=*string*] [, IDENTIFIER=*string*]

IDLitTool::RegisterOperation - Registers an operation component with the tool.

Obj->[IDLitTool::]RegisterOperation, *Name*, *ClassName*
[, ACCELERATOR=string] [, /CHECKED]
[, DESCRIPTION=string] [, /DROPLIST_EDIT]
[, DROPLIST_INDEX=value]
[, DROPLIST_ITEMS=string array] [, ICON=string]
[, IDENTIFIER=string] [, PROXY=string]
[, /SEPARATOR]

IDLitTool::RegisterStatusBarSegment - Registers a status message bar segment with the tool.

Obj->[IDLitTool::]RegisterStatusBarSegment, *Name*,
[, IDENTIFIER=*string*] [, NORMALIZED_WIDTH=*scalar*]

IDLitTool::RegisterVisualization - Registers a visualization component with the tool.

Obj->[IDLitTool::]RegisterVisualization, *Name*,
ClassName [, /DEFAULT] [, DESCRIPTION=*string*]
[, ICON=*string*] [, IDENTIFIER=*string*]
[, PROXY=*string*]

IDLitTool::SetProperty - Sets the value of an IDLitTool property

Obj->[IDLittool::]SetProperty[, *PROPERTY*=*value*]

IDLitTool::UnRegister - Unregisters a component with the tool.

Obj->[IDLitTool::]UnRegister, *Identifier*

IDLitTool::UnRegisterCustomization - Unregisters an operation class (that was previously registered as the graphics customization operation to be associated with this tool).

Obj->[IDLitTool::]UnRegisterCustomization

IDLitTool::UnRegisterFileReader - Unregisters a file reader component with the tool.

Obj->[IDLitTool::]UnRegisterFileReader, *Identifier*

IDLitTool::UnRegisterFileWriter - Unregisters a component with the tool.

Obj->[IDLitTool::]UnRegisterFileWriter, *Identifier*

IDLitTool::UnRegisterManipulator - Unregisters a manipulator component with the tool.

Obj->[IDLitTool::]UnRegisterManipulator, *Identifier*

IDLitTool::UnRegisterOperation - Unregisters an operation component with the tool.

Obj->[IDLitTool::]UnRegisterOperation, *Identifier*

IDLitTool::UnRegisterStatusBarSegment - Unregisters a status message bar segment.

Obj->[IDLitTool::]UnRegisterStatusBarSegment,
Identifier

IDLitTool::UnRegisterVisualization - Unregisters a visualization component with the tool.

Obj->[IDLitTool::]UnRegisterVisualization, *Identifier*

IDLitUI - A link between the underlying functionality of an iTool and the IDL widget interface.

Property: [, GROUP_LEADER=*WidgetID*]

IDLitUI::AddOnNofityObserver - Is used to register a specified iTool component object as wishing to receive messages generated by the DoOnNotify method of another specified iTool component object.

Obj->[IDLitUI::]AddOnNotifyObserver, *IdObserver*,
IdSubject

IDLitUI::Cleanup - Performs all cleanup on the object

OBJ_DESTROY, *Obj* or *Obj*->[IDLitUI::]Cleanup

IDLitUI::DoAction - Initiates an operation or action in the tool object associated with this user interface.

Result = *Obj*->[IDLitUI::]doAction(*Identifier*)

IDLitUI::GetProperty - Retrieves the value of an IDLitUI property.

Obj->[IDLitUI::]GetProperty, [*PROPERTY*=*variable*]

IDLitUI::GetTool - Returns an object reference to the iTool with which the user interface is associated.

Result = *Obj*->[IDLitUI::]GetTool()

IDLitUI::GetWidgetByName - returns the IDL Widget ID of a widget that has been registered with the user interface object via a call to the IDLitUI::RegisterWidget method.

Result = *Obj*->[IDLitUI::]GetWidgetByName(*Name*)

IDLitUI::Init - Initializes the IDLitUI object

Obj = OBJ_NEW('IDLitUI', *oTool*) or
Result = *Obj*->[IDLitUI::]Init(*oTool*)

IDLitUI::RegisterUIService - Registers a *user interface service* with the user interface.

Result = *Obj*->[IDLitUI::]RegisterUIService(*Name*,
Callback)

IDLitUI::RegisterWidget - Registers an IDL widget hierarchy with the user interface object.

Result = *Obj*->[IDLitUI::]RegisterWidget(*wID*, *Name*,
Callback[, /FLOATING])

IDLitUI::RemoveOnNotifyObserver - Is used to un-register a specified iTool component object as wishing to receive messages generated by the DoOnNotify method of another specified iTool component object.

Obj->[IDLitUI::]RemoveOnNotifyObserver, *IdObserver*,
IdSubject

IDLitUI::SetProperty - Sets the value of an IDLitUI property.

Obj->[IDLitUI::]SetProperty[, *PROPERTY*=*value*]

IDLitUI::UnRegisterUIService - Unregisters a user interface service with the user interface object.

Obj->[IDLitUI::]UnRegisterUIService, *Name*

IDLitUI::UnRegisterWidget - Unregisters a widget with the user interface object.

Obj->[IDLitUI::]UnRegisterWidget, *Name*

IDLitVisualization - The basis for all iTool visualizations. All visualization components subclass from this class.

Properties: [, CENTER_OF_ROTATION{Get, Init, Set}=[*x, y*] or [*x, y, z*]] [, /IMPACTS_RANGE{Get, Init, Set}] [, ISOTROPIC{Get, Init, Set}] [, /MANIPULATOR_TARGET{Get, Init, Set}] [, /PROPERTY_INTERSECTION{Get, Init}] [, TYPE{Init, Set}=*string or string array*]

IDLitVisualization::Add - Adds objects to the visualization container.

Obj->[IDLitVisualization::]Add, *Objects* [, /AGGREGATE] [, /NO_UPDATE] [, POSITION=*index*]

IDLitVisualization::Aggregate - Adds the given object(s) to this visualization's property aggregate.

Obj->[IDLitVisualization::]Aggregate, *Objects*

IDLitVisualization::BeginManipulation - Handles notifications that an IDLitManipulator object is about to manipulate this visualization.

Obj->[IDLitVisualization::]BeginManipulation, *Manipulator*

IDLitVisualization::Cleanup - Performs all cleanup on the object

OBJ_DESTROY, *Obj* or *Obj*->[IDLitVisualization::]Cleanup

IDLitVisualization::EndManipulation - Handles notifications that an IDLitManipulator object has finished manipulating this visualization.

Obj->[IDLitVisualization::]EndManipulation, *Manipulator*

IDLitVisualization::Get - Retrieves object(s) from the visualization.

Result = *Obj*->[IDLitVisualization::]Get([, /ALL] [, COUNT=*variable*] [, ISA=*string* or *array of strings*] [, POSITION=*index* or *array of indices*] [, /SKIP_PRIVATE])

IDLitVisualization::GetCenterRotation - Returns the center of rotation for this visualization.

Result = *Obj*->[IDLitVisualization::]GetCenterRotation ([, /NO_TRANSFORM] [, XRANGE=[*xmin, xmax*]] [, YRANGE=[*ymin, ymax*]] [, ZRANGE=[*zmin, zmax*]])

IDLitVisualization::GetCurrentSelectionVisual - Returns the currently active selection visual object for this visualization.

Result = *Obj*->[IDLitVisualization::] GetCurrentSelectionVisual()

IDLitVisualization::GetDataSpace - Returns a reference to the parent data space object within the graphics hierarchy that contains the visualization.

Result = *Obj*->[IDLitVisualization::]GetDataSpace([, /UNNORMALIZED])

IDLitVisualization::GetDataString - Retrieves a description of this visualization's data at the given *x, y,* and *z* location.

Result = *Obj*->[IDLitVisualization::]GetDataString(*XYZLocation*)

IDLitVisualization::GetDefaultSelectionVisual - Returns an object that serves as the default selection visual for this visualization.

Result = *Obj*->[IDLitVisualization::] GetDefaultSelectionVisual()

IDLitVisualization::GetManipulatorTarget - Retrieves the manipulator target associated with this visualization.

Result = *Obj*->[IDLitVisualization::] GetManipulatorTarget()

IDLitVisualization::GetProperty - Retrieves the value of a property or group of properties for the object.

Obj->[IDLitVisualization::]GetProperty [, *PROPERTY=variable*]

IDLitVisualization::GetRequestedAxesStyle - Returns the axes style requested by this visualization.

Result = *Obj*->[IDLitVisualization::] GetRequestedAxesStyle()

IDLitVisualization::GetSelectionVisual - Retrieves the selection visual for this visualization that corresponds to the given manipulator.

Result = *Obj*->[IDLitVisualization::]GetSelectionVisual(*Manipulator*)

IDLitVisualization::GetTypes - Identifies the types that this visualization represents, including base types and any specializations.

Result = *Obj*->[IDLitVisualization::]GetTypes()

IDLitVisualization::GetXYZRange - Computes the *x, y,* and *z* ranges of the overall contents of the visualization, taking into account the IMPACTS_RANGE property setting for itself and its contents.

Result = *Obj*->[IDLitVisualization::]GetXYZRange(*XRange*,*YRange*, *ZRange* [, /DATA] [, /NO_TRANSFORM])

IDLitVisualization::Init - Initializes the visualization object.

Obj = OBJ_NEW('IDLitVisualization' [, *PROPERTY=value*]) or *Result* = *Obj*->[IDLitVisualization::]Init([*PROPERTY=value*])

IDLitVisualization::Is3D - Determines whether or not this visualization (or any of its contents) is marked as being three-dimensional.

Result = *Obj*->[IDLitVisualization::]Is3D()

IDLitVisualization::IsIsotropic - Indicates whether or not this visualization (or any of its contents) is marked as being isotropic.

Result = *Obj*->[IDLitVisualization::]IsIsotropic()

IDLitVisualization::IsManipulatorTarget - Determines whether or not this visualization is a manipulator target.

Result = *Obj*->[IDLitVisualization::]IsManipulatorTarget()

IDLitVisualization::IsSelected - Determines if this visualization is currently selected or not.

Result = *Obj*->[IDLitVisualization::]IsSelected()

IDLitVisualization::Move - Changes the location of an object within the visualization container.

Obj->[IDLitVisualization::]Move, *Source, Destination*

IDLitVisualization::On2DRotate - Ensures that objects that use the visualization's data are notified when the rotation of the parent dataspace changes.

Obj->[IDLitVisualization::]On2DRotate, *Notifier, IsRotated*

IDLitVisualization::OnAxesRequestChange - Ensures that objects that use the visualization's data are notified when the axes of a contained object are changed.

Obj->[IDLitVisualization::]OnAxesRequestChange, *Notifier, AxesRequest*

IDLitVisualization::OnAxesStyleRequestChange - Ensures that objects that use the visualization's data are notified when the axes style of a contained object is changed.

Obj->[IDLitVisualization::]OnAxesStyleRequestChange, *Notifier, StyleRequest*

IDLitVisualization::OnDataChange - Ensures that objects that use the visualization's data are notified when the visualization's data changes.

Obj->[IDLitVisualization::]OnDataChange, *Notifier*

IDLitVisualization::OnDataComplete - Ensures that objects that use the visualization's data are notified when changes to the visualization's data are complete.

Obj->[IDLitVisualization::]OnDataComplete, *Notifier*

IDLitVisualization::OnDataRangeChange - Ensures that objects that use the visualization's data are notified when the range of the visualization's data changes.

Obj->[IDLitVisualization::]OnDataRangeChange, *Notifier*, *XRange*, *YRange*, *Zrange*

IDLitVisualization::OnDimensionChange - Ensures that objects that use the visualization's data are notified when the visualization's dimensionality changes.

Obj->[IDLitVisualization::]OnDimensionChange, *Notifier, Is3D*

IDLitVisualization::OnWorldDimensionChange - Ensures that objects that use the visualization's data are notified when the visualization's parent dataspace's dimensionality changes.

Obj->[IDLitVisualization::]OnWorldDimensionChange, *Notifier, Is3D*

IDLitVisualization::Remove - Removes the given object(s) from the visualization.

Obj->[IDLitVisualization::]Remove, *Object* [, /NO_UPDATE]

IDLitVisualization::RequestsAxes - Indicates whether or not the visualization requests axes. Returns a 1 if the visualization does request axes, or 0 if does not request axes.

Result = *Obj*->[IDLitVisualization::]RequestsAxes( )

IDLitVisualization::Restore - Performs any transitional work required after an object has been restored from a SAVE file.

Obj->[IDLitVisualization::]Restore

IDLitVisualization::Rotate - Rotates this visualization about the given axis by the given angle.

Obj->[IDLitVisualization::]Rotate, *Axis, Angle* [, CENTER_OF_ROTATION=[*x, y, z*]] [, /PREMULTIPLY]

IDLitVisualization::Scale - Scales the visualization by the given scale factors.

Obj->[IDLitVisualization::]Scale, *SX, SY, SZ* [, CENTER_OF_ROTATION=[*x, y*] | [*x, y, z*]] [, /PREMULTIPLY]

IDLitVisualization::Select - Handles notification of mechanisms that key off the current selection (such as the visualization browser) when this visualization has been selected.

Obj->[IDLitVisualization::]Select[, *Mode*] [, /ADDITIVE | /SELECT | /TOGGLE | /UNSELECT] [, /NO_NOTIFY]

DLitVisualization::Set3D - Sets a flag indicating this visualization is three-dimensional.

Obj->[IDLitVisualization::]Set3D, *Is3D* [, /ALWAYS] [, /AUTO_COMPUTE]

IDLitVisualization::SetAxesRequest - Sets the current axes request for this visualization.

Obj->[IDLitVisualization::]SetAxesRequest, *AxesRequest* [, /ALWAYS | /AUTO_COMPUTE] [, /NO_NOTIFY]

IDLitVisualization::SetAxesStyleRequest - Sets the current axes style request for this visualization.

Obj->[IDLitVisualization::]SetAxesStyleRequest, *StyleRequest* [, /NO_NOTIFY]

IDLitVisualization::SetCurrentSelectionVisual - Sets the current selection visual for the given manipulator.

Obj->[IDLitVisualization::]SetCurrentSelectionVisual, *Manipulator*

IDLitVisualization::SetData - Sets the data parameter of the visualization.

Result = *Obj*->[IDLitVisualization::]SetData(*Data*)

IDLitVisualization::SetDefaultSelectionVisual - Sets the default selection visual to be associated with this visualization.

Obj->[IDLitVisualization::]SetDefaultSelectionVisual, *SelectionVisual* [, POSITION=*value*]

IDLitVisualization::SetParameterSet - Associates a parameter set with this visualization.

Result = *Obj*->[IDLitVisualization::]SetParameterSet(*ParameterSet*)

IDLitVisualization::SetProperty - Sets the value of a property or group of properties for the object.

Obj->[IDLitVisualization::]SetProperty [, *PROPERTY*=*value*]

IDLitVisualization::UpdateSelectionVisual - Transforms this visualization's selection visual to match the visualization's geometry.

Obj->[IDLitVisualization::]UpdateSelectionVisual

IDLitVisualization::VisToWindow - Transforms given points from visualization data space to window device coordinates.

Obj->[IDLitVisualization::]VisToWindow, *InX*, *InY*, *InZ*, *OutX*, *OutY*, *OutZ* [, /NO_TRANSFORM] or *Obj*->[IDLitVisualization::] VisToWindow, *InX*, *InY*, *OutX*, *OutY* [, /NO_TRANSFORM] or *Obj*->[IDLitVisualization::] VisToWindow, *InVerts*, *OutVerts* [, /NO_TRANSFORM]

IDLitVisualization::WindowToVis - Transforms given points from window device coordinates to visualization data space.

Obj->[IDLitVisualization::]WindowToVis, *InX*, *InY*, *InZ*, *OutX*, *OutY*, *OutZ* or *Obj*->[IDLitVisualization::] WindowToVis, *InX*, *InY*, *OutX*, *OutY* or *Obj*->[IDLitVisualization::]WindowToVis, *InVerts*, *OutVerts*

IDLitWindow - The basis for all iTool visualization windows. All iTool visualization windows subclass from this class.

Properties: IDLitWindow inherits all properties from the IDLgrWindow superclass. See "IDLgrWindow" on page 99 for details.

IDLitWindow::Add - Adds the given object(s) to the window.

Obj->[IDLitWindow::]Add, *Objects* [, POSITION=*value*]

IDLitWindow::AddWindowEventObserver - Adds the given object(s) to the list of observers that are to be notified of events that occur within this window.

Obj->[IDLitWindow::]AddWindowEventObserver, *Objects*

IDLitWindow::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj*->[IDLitWindow::]Cleanup

IDLitWindow::ClearSelections - Clears the window's list of currently selected items (within its current view).

Obj->[IDLitWindow::]ClearSelections

IDLitWindow::DoHitTest - Performs a hit test to determine which visualizations within the destination are displayed at a given pixel location.

Result = *Obj*->DoHitTest(*X*, *Y* [, DIMENSIONS=[*width*, *height*]] [, /ORDER] [, SUB_HIT=*variable*] [, UNITS={0 | 1 | 2 | 3}])

IDLitWindow::GetEventMask - Returns a bitwise mask representing the events that are enabled for this window.

Result = *Obj*->[IDLitWindow::]GetEventMask([, BUTTON_EVENTS=*variable*] [, KEYBOARD_EVENTS=*variable*] [, /MOTION_EVENTS=*variable*] [, TIMER_EVENTS=*variable*] [, /TRACKING_EVENTS=*variable*])

IDLitWindow::GetProperty - Retrieves the value of an IDLitWindow property

Obj->[IDLitData::]GetProperty[, *PROPERTY*=*variable*]

IDLitWindow::GetSelectedItems - Returns the currently selected objects within this window's scene, which represents a container for all of the views (and their corresponding visualization hierarchies) that appear within a window.

Result = *Obj*->[IDLitWindow::]GetSelectedItems([, /ALL] [, COUNT=*named variable*])

IDLitWindow::Init - Initializes the window object.

Obj = OBJ_NEW('IDLitWindow'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLitWindow::]Init([, *PROPERTY*=*value*])

IDLitWindow::OnKeyboard - Handles notification (from the native window device) that a keyboard event has occurred, and passes along that notification to all observers in the list of window event observers.

Obj->[IDLitWindow::]OnKeyboard, *Window*, *IsASCII*, *Character*, *KeySymbol*, *X*, *Y*, *Press*, *Release*, *Modifiers*

IDLitWindow::OnMouseDown - Handles notification (from the native window device) that a mouse down event has occurred, and passes along that notification to all observers in the list of window event observers.

Obj->[IDLitWindow::]OnMouseDown, *Window*, *X*, *Y*, *ButtonMask*, *Modifiers*, *NumClicks*

IDLitWindow::OnMouseMotion - Handles notification (from the native window device) that a mouse motion event has occurred, and passes along that notification to all observers in the list of window event observers.

Obj->[IDLitWindow::]OnMouseMotion, *Window*, *X*, *Y*, *Modifiers*

IDLitWindow::OnMouseUp - Handles notification (from the native window device) that a mouse up event has occurred, and passes along that notification to all observers in the list of window event observers.

Obj->[IDLitWindow::]OnMouseUp, *Window*, *X*, *Y*, *ButtonMask*

IDLitWindow::OnScroll - Handles notification (from the native window device) that a scrolling event has occurred.

Obj->[IDLitWindow::]OnScroll, *X*, *Y*

IDLitWindow::OnTimer - handles notification (from the native window device) that a timer event has occurred, and passes that notification to all observers in the list of window event observers by calling each observer's OnTimer method.

Obj->[IDLitWindow::]OnTimer

IDLitWindow::Remove - Removes the given object(s) from the window.

Obj->[IDLitWindow::]Remove, *Object* [, /ALL] [, POSITION=index]

IDLitWindow::RemoveWindowEventObserver - Removes the given object(s) from the list of observers that are notified of events that occur within this window.

Obj->[IDLitWindow::]RemoveWindowEventObserver, *Objects*

IDLitWindow::SetCurrentZoom - Sets the current zoom factor for this window by changing its virtual dimensions.

Obj->[IDLitWindow::]SetCurrentZoom, *ZoomFactor* [, /RESET]

IDLitWindow::SetEventMask - Enables the given events within this window.

Obj->[IDLitWindow::]SetEventMask([*EventMask*] [, /BUTTON_EVENTS] [, /KEYBOARD_EVENTS] [, /MOTION_EVENTS] [, /TIMER_EVENTS] [, /TRACKING_EVENTS])

IDLitWindow::SetManipulatorManager - Sets the given IDLitManipulatorManager object as the current manager of this window's manipulators.

Obj->[IDLitWindow::]SetManipulatorManager, *Manager*

IDLitWindow::SetProperty - Sets the value of an IDLitWindow property.

Obj->[IDLitWindow::]SetProperty[, *PROPERTY*=*value*]

IDLitWindow::SetTimerInterval - Specifies the floating-point number of seconds between timer events.

Obj->[IDLitWindow::]SetProperty[, *PROPERTY*=*value*]

IDLitWindow::ZoomIn - Causes the current zoom factor for this window to be increased (that is, multiplied by the factor given by the window's ZOOM_BASE property).

Obj->[IDLitWindow::]ZoomIn

IDLitWindow::ZoomOut - Causes the current zoom factor for this window to be decreased (that is, divided by the factor given by the window's ZOOM_BASE property).

Obj->[IDLitWindow::]ZoomOut

IDLitWriter - The definition of the interface and the process used to construct file writers for the iTools framework. When a new file writer is constructed for the iTools system, a new class is subclassed from this IDLitWriter class.

IDLitWriter::Cleanup - Performs all cleanup on the object

OBJ_DESTROY, *Obj* or *Obj*->[IDLitWriter::]Cleanup

IDLitWriter::GetFileExtensions - Is called by the system to retrieve the file extensions supported by this particular writer.

Result = *Obj*->[IDLitWriter::]GetFileExtensions([COUNT=*count*])

IDLitWriter::GetFilename - Is called by the system to retrieve the current filename associated with this writer.

Result = *Obj*->[IDLitWriter::]GetFilename()

IDLitWriter::GetProperty - Retrieves the value of an IDLitWriter property.

Obj->[IDLitWriter::]GetProperty [, *PROPERTY*=*variable*]

IDLitWriter::Init - Initializes the IDLitWriter object

Obj = OBJ_NEW('IDLitWriter', [, *Extensions*] [, *PROPERTY*=*value*]) or

Result = *Obj*->[IDLitWriter::]Init([, *Extensions*] [, *PROPERTY*=*value*])

IDLitWriter::IsA - Is called by the system to determine if the given file is of the type supported by this file writer.

Result = *Obj*->[IDLitWriter::]IsA(*Filename*)

IDLitWriter::SetData - Is called by the system to set the data for the current file.

Result = *Obj*->[IDLitWriter::]SetData(*Data*)

IDLitWriter::SetFilename - Is called by the system to set the current filename associated with this writer.

Obj->[IDLitWriter::]SetFilename, *Filename*

IDLitWriter::SetProperty - Sets the value of an IDLitWriter property

Obj->[IDLitWriter::]SetProperty[, *PROPERTY*=*value*]

IDLjavaObject - An IDL object encapsulating a Java object. IDL provides data type and other translation services, allowing IDL programs to access the Java object's methods and properties using standard IDL syntax.

IDLjavaObject::GetProperty - Retrieves properties (known as data members in Java) from the Java object that underlies the IDLjavaObject.

Obj->[IDLjavaObject::]GetProperty [, *PROPERTY*=*variable*]

IDLjavaObject::Init - Instantiates the given Java object and establishes a link between the resulting IDL object with the underlying Java object.

Obj = OBJ_NEW('IDLjavaObject$*JAVACLASSNAME*', *JavaClassName*[, *Arg1*, ...]) or

Result = *Obj*->[IDLjavaObject$*JAVACLASSNAME*::]Init(*JavaClassName* [, *Arg1*, ...])

IDLjavaObject::SetProperty - Sets properties (known as data members in Java) for the Java object that underlies an instance of IDLjavaObject.

Obj->[IDLjavaObject::]SetProperty [, *PROPERTY*=*value*]

TrackBall - Translates widget events from a draw widget (created with the WIDGET_DRAW function) into transformations that emulate a virtual trackball (for transforming object graphics in three dimensions).

Properties: [, AXIS{Init}={0 | 1 | 2}] [, /CONSTRAIN{Init}] [, MOUSE{Init}={1 | 2 | 4}]

TrackBall::Init - Initializes the TrackBall object.

Obj = OBJ_NEW('TrackBall', *Center*, *Radius* [, *PROPERTY*=*value*]) or

Result = *Obj*->[TrackBall::]Init(*Center*, *Radius* [, *PROPERTY*=*value*])

Trackball::Reset - Resets the state of the TrackBall object.

Obj->[TrackBall::]Reset, *Center*, *Radius* [, AXIS={0 | 1 | 2}] [, /CONSTRAIN] [, MOUSE={1 | 2 | 4}]

TrackBall::Update - Updates the state of the TrackBall object based on the information contained in the input widget event structure.

Result = *Obj*->[TrackBall::]Update(*sEvent* [, MOUSE={1 | 2 | 4}] [, TRANSFORM=*variable*] [, /TRANSLATE])

Statements

Assignment

***variable* = *expression* -** Assigns a value to a variable.

***variable*[*subscripts*] = *expression* -** Assigns a value to the elements of an array specified by the array subscripts.

***variable*[*subscript_range*] = *expression* -** Assigns a value to the elements of an array specified by the array subscript range.

Program Control

Compound Statements

BEGIN...END - Defines a block of statements.

BEGIN
statements
END | ENDIF | ENDELSE | ENDFOR | ENDREP | ENDWHILE

Conditional Statements

IF...THEN...ELSE - Conditionally executes a statement or block of statements.

IF *expression* THEN *statement* [ELSE *statement*]
or
IF *expression* THEN BEGIN
statements
ENDIF [ELSE BEGIN
statements
ENDELSE]

CASE - Selects one statement for execution from multiple choices, depending on the value of an expression.

CASE *expression* OF
expression: *statement*
...
expression: *statement*
[ELSE: *statement*]
ENDCASE

SWITCH - Selects one statement for execution from multiple choices, depending upon the value of an expression.

SWITCH *expression* OF
expression: *statement*
...
expression: *statement*
[ELSE: *statement*]
ENDSWITCH

Loop Statements

FOR...DO - Executes one or more statements repeatedly, while incrementing or decrementing a variable with each repetition, until a condition is met.

FOR *Variable* = *Init*, *Limit* [, *Increment*] DO *statement*
or
FOR *Variable* = *Init*, *Limit* [, *Increment*] DO BEGIN
statements
ENDFOR

REPEAT...UNTIL - Repeats statement(s) until expression evaluates to true. Subject is always executed at least once.

REPEAT *statement* UNTIL *expression*
or
REPEAT BEGIN
statements
ENDREP UNTIL *expression*

WHILE...DO - Performs statement(s) as long as expression evaluates to true. Subject is never executed if condition is initially false.

WHILE *expression* DO *statement*
or
WHILE *expression* DO BEGIN
statements
ENDWHILE

Jump Statements

BREAK - Immediately exits from a loop (FOR, WHILE, REPEAT), CASE, or SWITCH statement without resorting to GOTO statements.

BREAK

CONTINUE - Immediately starts the next iteration of the enclosing FOR, WHILE, or REPEAT loop.

CONTINUE

GOTO - Transfers program control to point specified by *label*.

GOTO, *label*

Functions and Procedures

COMPILE_OPT - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

COMPILE_OPT opt_1 [, opt_2, ..., opt_n]

Note: opt_n can be IDL2, DEFINT32, HIDDEN, LOGICAL_PREDICATE, OBSOLETE, STRICTARR, or STRICTARRSUBS

FORWARD_FUNCTION - Causes argument(s) to be interpreted as functions rather than variables (versions of IDL prior to 5.0 used parentheses to declare arrays).

FORWARD_FUNCTION $Name_1$, $Name_2$, ..., $Name_n$

FUNCTION - Defines a function.

FUNCTION *Function_Name*, $parameter_1$, ..., $parameter_n$

PRO - Defines a procedure.

PRO *Procedure_Name*, $argument_1$, ..., $argument_n$

***Procedure_Name* -** Calls a procedure.

Procedure_Name, $argument_1$, ..., $argument_n$

***Result* = *FUNCTION*(arg_1, ..., arg_n) -** Calls a function.

Variable Scope

COMMON - Creates a common block.

COMMON *Block_Name*, $Variable_1$, ..., $Variable_n$

Executive Commands

Executive commands must be entered at the IDL command prompt. They cannot be used in programs.

.COMPILE - Compiles programs without running.

.COMPILE [*File*$_1$,..., *File*$_n$]

To compile from a temporary file: .COMPILE -f *File TempFile*

.CONTINUE - Continues execution of a stopped program.

.CONTINUE

.EDIT - Opens files in editor windows of the IDLDE (Windows and Motif only). Note that filenames are separated by spaces, not commas.

.EDIT *File*$_1$ [*File*$_2$ *File*$_n$]

.FULL_RESET_SESSION - Does everything .RESET_SESSION does, plus additional reset tasks such as unloading sharable libraries.

.FULL_RESET_SESSION

.GO - Executes previously-compiled main program.

.GO

.OUT - Continues execution until the current routine returns.

.OUT

.RESET_SESSION - Resets much of the state of an IDL session without requiring the user to exit and restart the IDL session.

.RESET_SESSION

.RETURN - Continues execution until RETURN statement.

.RETURN

.RNEW - Erases main program variables and then does .RUN.

.RNEW [*File*$_1$,...,*File*$_n$]

To save listing in a file:.RNEW -L *ListFile.lis File*$_1$ [, *File*$_2$,..., *File*$_n$]

To display listing on screen: .RNEW -T *File*$_1$ [, *File*$_2$,..., *File*$_n$]

.RUN - Compiles and executes IDL commands from files or keyboard.

.RUN [*File*$_1$,..., *File*$_n$]

To save listing in a file:.RUN -L *ListFile.lis File*$_1$ [, *File*$_2$,..., *File*$_n$]

To display listing on screen: .RUN -T *File*$_1$ [, *File*$_2$,..., *File*$_n$]

.SKIP - Skips over the next *n* statements and then single steps.

.SKIP [*n*]

.STEP - Executes one or *n* statements from the current position.

.STEP [*n*] or .S [*n*]

.STEPOVER - Executes a single statement if the statement doesn't call a routine.

.STEPOVER [*n*] or .SO [*n*]

.TRACE - Similar to .CONTINUE, but displays each line of code before execution.

.TRACE

Special Characters

The following table lists the characters that have a special meaning in IDL:

Character	Description
Ampersand (&)	•Separates multiple commands on a single line
Apostrophe (')	•Delimits string constants •Indicates part of octal or hex constant
Asterisk (*)	•Multiplication operator •Array subscript range •Pointer dereference (if in front of a valid pointer)
At Sign (@)	•Include character: Used at beginning of a line to cause the IDL compiler to substitute the contents of the file whose name appears after the @ symbol for the line. •In interactive mode, @ is used to execute a batch file.
Colon (:)	•Ends label identifiers •Separates start and end subscript ranges
Dollar Sign ($)	•Continue current command on the next line •Issue operating system command if entered on a line by itself
Exclamation Point (!)	•First character of system variable names and font-positioning commands
Period (.)	•First character of executive commands •Indicates floating-point numbers •Indicates fields in a structure, such as in mystructure.field1
Question Mark (?)	•Invokes online help when entered at the IDL command line •Part of the ?: ternary operator used in conditional expressions
Semicolon (;)	•First character of comment field. Everything after the semicolon is ignored by IDL. Semicolon can be used as the first character or after an IDL command: ; This is a comment COUNT = 5 ; Set variable COUNT to 5

Subscripts

Subscripts are used to designate array elements to receive new values, and to retrieve the value of one or more array elements. IDL arrays are zero-based, meaning the first element is element 0.

Example	Description
Vector[**i**]	Element i + 1 of a vector. Vector[12] denotes the value of the 13th element of Vector.
Array[**i**, **j**]	The element stored at column i, row j of an array.
Vector[**i:j**]	Elements i through j of a vector.
Vector[**i:***]	Elements from i through the end of a vector.
Array[**i**, ***]	Column i of a two-dimensional array.
Array[***, **j**]	The jth row of a two-dimensional array.
Array[**i:j**, **m:n**]	Subarray of columns i though j, rows m through n.
Array[**i:j**:k]	A range of subscripts, written [k_0:k_1:k_2], denoting every k_2th element within the range of subscripts k_0 through k_1 (k_0 must not be greater than k_1). k_2 is referred to as the subscript *stride*.
Array[**i:*:k**]	Every k_2th element from a given element to the last element of the dimension, written as [k_0:*:k_2]. k_2 is referred to as the subscript *stride*.
Array[**Array2**]	The elements of Array whose subscripts are the values of Array2.
(Array_Expression)[**i**]	Element i of an array-valued expression.

Operators

Mathematical Operators

+ Addition, String Concatenation
-. Subtraction and Negation
***.** Multiplication, Pointer dereference
/. Division
^. Exponentiation
++. Increment
--. Decrement
MOD. Modulo

Minimum/Maximum Operators

< The Minimum Operator
>. The Maximum Operator

Matrix Operators

 # and ## Matrix Multiplication

Logical Operators

&& - Logical AND
||. Logical OR
~. Logical NOT
AND. Bitwise AND
NOT. Bitwise complement
OR. Bitwise OR
XOR. Bitwise exclusive OR

Relational Operators

EQ - Equal to
GE. Greater than or equal to
GT. Greater than
LE. Less than or equal to
LT. Less than
NE. Not equal to

Other Operators

[ ] Array concatenation, enclose array subscripts
( ) Group expressions to control order of evaluation
= Assignment
***op*=** Compound assignment
?: Conditional expression
-> Object method invocation

Operator Precedence

Operators with the highest precedence are evaluated first. Operators with equal precedence are evaluated from left to right.

Priority	Operator
First (highest)	( ) (parentheses, to group expressions)
	[ ] (brackets, to concatenate arrays)
Second	. (structure field dereference)
	[ ] (brackets, to subscript an array)
	( ) (parentheses, used in a function call)
Third	* (pointer dereference)
	^ (exponentiation)
	++ (increment)
	-- (decrement)
Fourth	* (multiplication)
	# and ## (matrix multiplication)
	/ (division)
	MOD (modulus)
Fifth	+ (addition)
	- (subtraction and negation)
	< (minimum)
	> (maximum)
	NOT (bitwise negation)
Sixth	EQ (equality)
	NE (not equal)
	LE (less than or equal)
	LT (less than)
	GE (greater than or equal)
	GT (greater than)
Seventh	AND (bitwise AND)
	OR (bitwise OR)
	XOR (bitwise exclusive OR)
Eighth	&& (logical AND)
	|| (logical OR)
	~ (logical negation)
Ninth	?: (conditional expression)

System Variables

IDL system variables contain useful constants, control plotting defaults, and store information about the current IDL session.

Constant System Variables

!DPI - Double-precision pi (p).

!DTOR - Degrees to radians, pi/180 ≈ 0.01745.

!MAP - Read-only system variable used by MAP_SET.

!PI - Single-precision pi (p).

!RADEG - Radians to degrees, 180/pi ≈ 57.2958.

!VALUES - Single- and double-precision NaN and Infinity values.

Graphics System Variables

!D - Information about current graphics device.

Fields: FILL_DIST - line interval, in device coordinates
FLAGS - longword of flags
N_COLORS - number of simultaneously available colors
NAME - string containing name of device
ORIGIN - pan/scroll offset (*pan, scroll*)
TABLE_SIZE - number of color table indices
UNIT - logical number of file open for output
WINDOW - index of currently open window
X_CH_SIZE, Y_CHAR_SIZE - width/height of rectangle that encloses the average character in current font, in device units (usually pixels)
X_PX_CM, Y_PX_CM - approx. number of pixels/cm
X_SIZE, Y_SIZE - total size of the display or window, in device units
X_VSIZE, Y_VSIZE - size of visible area of display or window
ZOOM - X and Y zoom factors

!ORDER - Direction of image transfer: 0=bottom up, 1=top down.

!P - Information for plotting procedures.

Fields:
BACKGROUND - background color index
CHANNEL - default source or destination channel
CHARSIZE - character size of annotation when Hershey fonts are selected
CHARTHICK - integer specifying thickness of vector fonts
CLIP - device coords of clipping window ([x_0, y_0, z_0), (x_1, y_1, z_1)]
COLOR - default color index
FONT - integer specifying graphics text font system to use (-1 for Hershey, 0 for output device font, 1 for TrueType)
LINESTYLE - style of lines that connect points (see "Line Styles" on page 124)
MULTI - integer array: [*plots remaining on page, columns per page, rows per page, plots in Z direction, 0 for left to right or 1 for top to bottom*]
NOCLIP - if set, inhibits clipping of graphic vectors
NOERASE - set to nonzero value to prevent erasing
NSUM - number of adjacent points to average
POSITION - normalized coords of plot window (*x0, y0, x1, y1*)
PSYM - plotting symbol index (see "Plotting Symbols" on page 124)
REGION - normalized coords of plot region (*x0, y0, x1, y1*)
SUBTITLE - plot subtitle (under X axis label)
T - homogeneous 4 x 4 transformation matrix
T3D - enables 3D to 2D transformation
THICK - thickness of lines connecting points
TITLE - main plot title
TICKLEN - tick mark length (0.0 to 1.0)

!X, !Y, !Z - Axis structures for X, Y, and Z axes.

Fields: CHARSIZE - character size of annotation when Hershey fonts are selected
CRANGE - output axis range
GRIDSTYLE - linestyle for tick marks/grids (see "Line Styles" on page 124)
MARGIN - 2-element array specifying plot window margins, in units of char size ([*left or bottom, right or top*])
MINOR - number of minor tick marks
OMARGIN - 2-element array specifying plot window outer margins, in units of char size ([*left or bottom, right or top*])
RANGE - 2-element vector specifying input axis range (*min, max*)
REGION - normalized coords of region (2-element floating-point array)
S - 2-element array specifying scaling factors for conversion between data and normalized coords
STYLE - style of the axis encoded as bits in a longword. 1=exact, 2=extend, 4=no axis, 8=no box, 16=inhibit setting Y axis min to 0 when data are all greater than 0 (add values together for multiple effects)
THICK - thickness of axis line
TICKFORMAT - format string or string containing name of function that returns format string used to format axis tick mark labels

!X, !Y, !Z - *continued*

TICKINTERVAL - indicates the interval between major tick marks for the first axis level

TICKLAYOUT - indicates the tick layout style to be used to draw each level of the axis

TICKLEN - tick mark length, in normal coords

TICKNAME - annotation for each tick (string array)

TICKS - number of major tick intervals

TICKUNITS - indicates the units to be used for axis tick labeling

TICKV - data values for each tick mark (array)

TITLE - string containing axis title

TYPE - type of axis (0 for linear, 1 for logarithmic)

WINDOW - normalized coords of axis end points (2-element floating-point array

Error Handling/Informational System Variables

!ERROR_STATE - Structure containing all error information.

Fields: NAME - string containing error name of IDL-generated component of last error message (read-only).

BLOCK - string containing name of message block for IDL-generated component of last error message (read-only).

CODE - long-integer containing error code of IDL-generated component of last error message.

SYS_CODE - long-integer containing error code of operating system component of last error message.

SYS_CODE_TYPE - A string describing the type of system code contained in SYS_CODE.

MSG - string containing text of IDL-generated component of last error message (read-only).

MSG_PREFIX - string containing prefix string used for error messages.

SYS_MSG - string containing text of operating system generated component of last error message (read-only).

!EXCEPT - Controls when IDL checks for math error conditions (0=never report exceptions, 1=report exceptions when interpreter is returning to interactive prompt, 2=report exceptions at end of each IDL statement).

!MOUSE - Status from the last cursor read operation.

Fields: X, Y - location (in device coords) of cursor when mouse button was pressed

BUTTON - specifies which mouse button was pressed (1 if left, 2 if middle, 4 if right) TIME - number of milliseconds since a base time

!WARN - Report use of obsolete routines.

Fields: OBS_ROUTINES - if set to 1, IDL generates warnings when it encounters use of obsolete routines

OBS_SYSVARS - if set to 1, IDL generates warnings when it encounters use of obsolete system variables

PARENS - if set to 1, IDL generates warnings when it encounters use parentheses to index array

IDL Environment System Variables

!CPU - Read-only variable that supplies information about the state of the system processor, and of IDL's use of it.

Fields: HW_VECTOR - True (1) if the system supports a vector unit (e.g. Macintosh Altivec/Velocity Engine) or False (0) otherwise.

VECTOR_ENABLE - True (1) if IDL will use a vector unit, if such a unit is available on the current system, and False (0) otherwise.

HW_NCPU - The number of CPUs contained in the system on which IDL is currently running.

TPOOL_NTHREADS - The number of threads that IDL will use in thread pool computations.

TPOOL_MIN_ELTS - The number of elements in a computation that are necessary before IDL will use the thread pool to perform the work

TPOOL_MAX_ELTS - The maximum number of elements in a computation for which IDL will use the thread pool.

!DIR - Location of the main IDL directory.

!DLM_PATH - Indicates where IDL looks for Dynamically Loadable Modules when started. Read-only.

!EDIT_INPUT - Enables/disables keyboard line editing.

!HELP_PATH - Lists directories IDL will search for online help files

!JOURNAL - Logical unit number of journal output, or 0.

!MORE - Set to 0 to prevent paginating help text.

!MAKE_DLL - Used to configure how IDL uses the CALL_EXTERNAL, DLMs, and LINKIMAGE for the current platform.

!PATH - Search path for IDL routines.

UNIX: colon-separated list of directories.

Windows: semicolon-separated list of directories.

!PROMPT - String to be used for IDL prompt.

!QUIET - Suppresses informational messages if set to nonzero.

!VERSION - Type, architecture, and version of IDL.

Fields: ARCH - CPU hardware architecture of the system.

OS - The name of the underlying operating system kernel e.g. AIX, sunos, Win32).

OS_FAMILY - The generic name of the operating system (e.g. UNIX, Windows).

OS_NAME - The vendor's name for the operating environment (e.g. Solaris, Microsoft Windows).

RELEASE - The IDL version number.

BUILD_DATE - Date the IDL exeuctable was compiled.

Graphics Information

Direct Graphics Devices

CGM - The CGM Device
HP - The HP-GL Device
NULL - The Null Display Device
PCL - The PCL Device
PRINTER - The Printer Device
PS - The PostScript Device
REGIS - The Regis Terminal Device
TEK - The Tektronix Device
WIN - The Microsoft Windows Device
X - The X Windows Device
Z - The Z-Buffer Device

Graphics Keywords

The following keywords are used with IDL plotting routines (AXIS, CONTOUR, PLOT, OPLOT, SHADE_SURF, and SURFACE) and graphics routines (CURSOR, ERASE, PLOTS, POLYFILL, TV, TVCRS, TVRD, and XYOUTS). Many have system variable equivalents. Not all keywords work with all routines. Listings such as {XYZ}KEYWORD indicate that there are 3 keywords, one for each axis (e.g., XCHARSIZE, YCHARSIZE, ZCHARSIZE).

BACKGROUND - Background color index when erasing.
CHANNEL - Channel index or mask for multi-channel displays.
CHARSIZE - Overall character size.
{XYZ}CHARSIZE - Character size for axes.
CHARTHICK - Overall thickness for vector fonts.
CLIP - Coordinates of clipping window.
COLOR - Color index for data, text, line, or polygon fill.
DATA - Set to plot in data coordinates.
DEVICE - Set to plot in device coordinates.
FONT - Text font index: -1 for vector, 0 for hardware fonts.
{XYZ}GRIDSTYLE - Linestyle index for tickmarks and grids.
LINESTYLE - LInestyle used to connect data points.
{XYZ}MARGIN - Margin of plot window in character units.
{XYZ}MINOR - number of minor tick marks.
NOCLIP - Set to disable clipping of plot.
NODATA - Set to plot only axes, titles, and annotation w/o data.
NOERASE - Set to inhibit erasing before new plot.
NORMAL - Set to plot in normal coordinates.
ORIENTATION - Angle (in degrees counter-clockwise) for text.
POSITION - Position of plot window.
PSYM - Use plotting symbols to plot data points.
{XYZ}RANGE - Axis range.
{XYZ}STYLE - Axis type.
SUBTITLE - String for subtitle.
SYMSIZE - Size of PSYM plotting symbols.
T3D - Set to use 3D transformation store in !P.T.
THICK - Overall line thickness.
{XYZ}THICK - Thickness of axis and tickmark lines.
{XYZ}TICKFORMAT - Allows advanced formatting of tick labels.
{XYZ}TICKINTERVAL - Set to indicate the interval between major tick marks for the first axis level.
{XYZ}TICKLAYOUT - Set to indicate the tick layout style to be used to draw each level of the axes.
TICKLEN - Length of tickmarks in normal coordinates. 1.0 produces a grid. Negative values extend outside window.
{XYZ}TICKLEN - Tickmark lengths for individual axes.
{XYZ}TICKNAME - String array of up to 30 labels for tickmark annotation.
{XYZ}TICKS - Number of major tick intervals for axes.
{XYZ}TICKUNITS - Set to indicate the units to be used for axis tick labeling.
{XYZ}TICKV - Array of up to 30 elements for tick mark values.
{XYZ}TICK_GET - Variable in which to return values of tick marks.
TITLE - String for plot title.
{XYZ}TITLE - String for specified axis title.
ZVALUE - The Z coordinate for a 2D plot in 3D space.
Z - Z coordinate if Z argument not specified in 3D plot call.

Line Styles

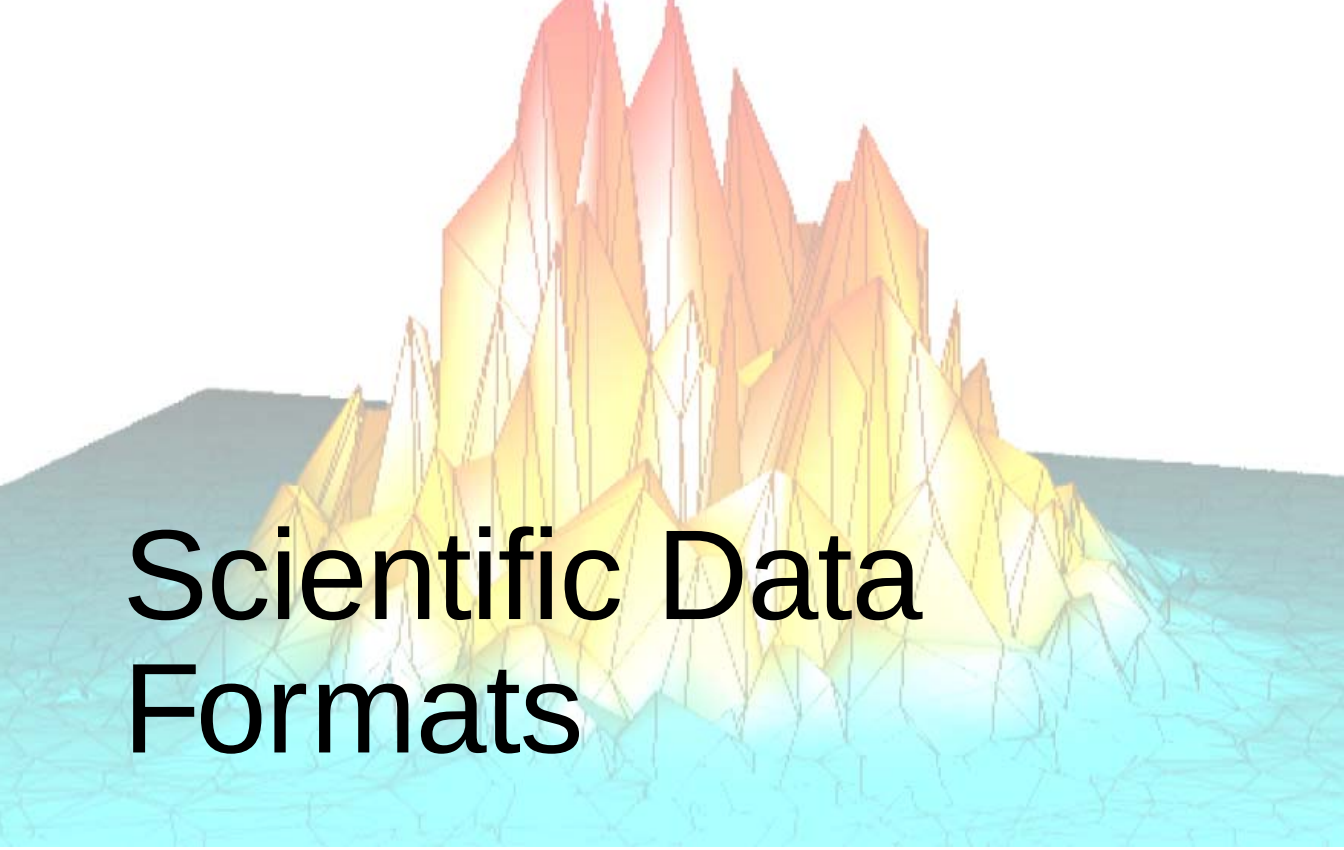
The LINESTYLE keyword to the Direct Graphics plotting routines OPLOT, PLOT, PLOTS, and SURFACE accepts the following values:

Index	Linestyle
0	Solid
1	Dotted
2	Dashed
3	Dash Dot
4	Dash Dot Dot Dot
5	Long Dashes

Plotting Symbols

The PSYM keyword to Direct Graphics plotting routines OPLOT, PLOT, and PLOTS accepts the following values:

PSYM Value	Plotting Symbol
1	Plus sign (+)
2	Asterisk (*)
3	Period (.)
4	Diamond
5	Triangle
6	Square
7	X
8	User-defined. See USERSYM procedure.
9	Undefined
10	Histogram mode.



Scientific Data Formats

This quick reference guide contains an alphabetical listing of all scientific data format routines including CDF*, EOS* and HDF* routines. The alphabetical listing contains all functions, procedures, and statements including the syntax of each.

CDF Routines

CDF_ATTCREATE - Creates a new attribute.
Result = CDF_ATTCREATE(*Id*, *Attribute_Name*
[, /GLOBAL_SCOPE] [, /VARIABLE_SCOPE])

CDF_ATTDELETE - Deletes attribute from specified CDF file.
CDF_ATTDELETE, *Id*, *Attribute* [, *EntryNum*]
[, /ZVARIABLE]

CDF_ATTEXISTS - Determines whether specified attribute exists.
Result = CDF_ATTEXISTS(*Id*, *Attribute* [, *EntryNum*]
[, /ZVARIABLE])

CDF_ATTGET - Reads an attribute entry from a CDF file.
CDF_ATTGET, *Id*, *Attribute*, *EntryNum*, *Value*
[, CDF_TYPE= *variable*] [, /ZVARIABLE]

CDF_ATTINQ - Obtains information about specified attribute.
CDF_ATTINQ, *Id*, *Attribute*, *Name*, *Scope*, *MaxEntry*
[, *MaxZEntry*]

CDF_ATTNUM - Returns an attribute number.
Result = CDF_ATTNUM(*Id*, *Attribute_Name*)

CDF_ATTPUT - Writes an attribute entry to a CDF file.
CDF_ATTPUT, *Id*, *Attribute*, *EntryNum*, *Value*
[, /ZVARIABLE]

CDF_ATTRENAME - Renames an existing attribute.
CDF_ATTRENAME, *Id*, *OldAttr*, *NewName*

CDF_CLOSE - Closes specified Common Data Format file.
CDF_CLOSE, *Id*

CDF_COMPRESSION - Sets or returns the compression mode for a CDF file and/or variables.
CDF_COMPRESSION, *Id*
[, GET_COMPRESSION=*variable*]
[, GET_GZIP_LEVEL=*variable*]
[, GET_VAR_COMPRESSION=*variable*]
[, GET_VAR_GZIP_LEVEL=*variable*]
[, SET_COMPRESSION={0 | 1 | 2 | 3 | 5}]
[, SET_GZIP_LEVEL=*integer*{1 to 9}]
[, SET_VAR_COMPRESSION={0 | 1 | 2 | 3 | 5}]
[, SET_VAR_GZIP_LEVEL=*integer*{1 to 9}]
[, VARIABLE=*variable name or index*]
[, /ZVARIABLE]

CDF_CONTROL - Obtains or sets information for a CDF file.
CDF_CONTROL, *Id* [, ATTRIBUTE=*name or number*]
[, GET_ATTR_INFO=*variable*]
[, GET_CACHESIZE=*variable*]
[, GET_COPYRIGHT=*variable*]
[, GET_FILENAME=*variable*]
[, GET_FORMAT=*variable*]
[, GET_NEGTOPOSFP0_MODE=*variable*]
[, GET_NUMATTRS=*variable*]
[, GET_READONLY_MODE=*variable*]
[, GET_RVAR_CACHESIZE=*variable*]
[, GET_VAR_INFO=*variable*]
[, GET_ZMODE=*variable*]
[, GET_ZVAR_CACHESIZE=*variable*]
[, SET_CACHESIZE=*value*]
[, SET_EXTENDRECS=*records*]
[, SET_INITIALRECS=*records*]
[, /SET_NEGTOPOSFP0_MODE]
[, SET_PADVALUE=*value*]
[, /SET_READONLY_MODE]
[, SET_RVAR_CACHESIZE=*value*{See Note}]
[, SET_RVARS_CACHESIZE=*value*{See Note}]
[, SET_ZMODE={0 | 1 | 2}]
[, SET_ZVAR_CACHESIZE=*value*{See Note}]
[, SET_ZVARS_CACHESIZE=*value*{See Note}]
[, VARIABLE=*name or index*] [, /ZVARIABLE]
Note: Use only with MULTI_FILE CDF files

CDF_CREATE - Creates a new Common Data Format file.
Result = CDF_CREATE(*Filename*, [*Dimensions*]
[, /CLOBBER] [, /MULTI_FILE | , /SINGLE_FILE]
[, /COL_MAJOR | , /ROW_MAJOR])
Encoding Keywords (pick one):
[, /ALPHAOSF1_ENCODING]
[, /ALPHAVMSD_ENCODING]
[, /ALPHAVMSG_ENCODING]
[, /DECSTATION_ENCODING]
[, /HOST_ENCODING]
[, /HP_ENCODING] [, /IBMRS_ENCODING]
[, /IBMPC_ENCODING] [, /MAC_ENCODING]
[, /NETWORK_ENCODING] [, /NEXT_ENCODING]
[, /SGI_ENCODING] [, /SUN_ENCODING]
Decoding Keywords (pick one):
[, /ALPHAOSF1_DECODING]
[, /ALPHAVMSD_DECODING]
[, /ALPHAVMSG_DECODING]
[, /DECSTATION_DECODING]
[, /HOST_DECODING] [, /HP_DECODING]
[, /IBMRS_DECODING] [, /IBMPC_DECODING]
[, /MAC_DECODING] [, /NETWORK_DECODING]
[, /NEXT_DECODING] [, /SGI_DECODING]
[, /SUN_DECODING]

CDF_DELETE - Deletes specified Common Data Format file.
CDF_DELETE, *Id*

CDF_DOC - Gets documentation information about a CDF file.
CDF_DOC, *Id*, *Version*, *Release*, *Copyright* [, INCREMENT=*variable*]

CDF_ENCODE_EPOCH - Encodes CDF_EPOCH variable into a string.
Result = CDF_ENCODE_EPOCH(*Epoch* [, EPOCH={0 | 1 | 2 | 3}])

CDF_EPOCH - Computes/breaks down CDF_EPOCH values.
CDF_EPOCH, *Epoch*, *Year* [, *Month*, *Day*, *Hour*, *Minute*, *Second*, *Milli*] [, /BREAKDOWN_EPOCH] [, /COMPUTE_EPOCH]

CDF_ERROR - Returns explanation of a given status code.
Result = CDF_ERROR(*Status*)

CDF_EXISTS - Returns True if CDF data format library is supported on the current IDL platform.
Result = CDF_EXISTS( )

CDF_INQUIRE - Returns global information about CDF file.
Result = CDF_INQUIRE(*Id*)

CDF_LIB_INFO - Returns information about the CDF Library being used.
CDF_LIB_INFO [, COPYRIGHT=*variable*] [, INCREMENT=*variable*] [, RELEASE=*variable*] [, SUBINCREMENT=*variable*] [, VERSION=*variable*]

CDF_OPEN - Opens an existing Common Data Format file.
Result = CDF_OPEN(*Filename*)

CDF_PARSE_EPOCH - Parses input string into a double precision value properly formatted for use as CDF_EPOCH variable.
Result = CDF_PARSE_EPOCH(*Epoch_string*)

CDF_VARCREATE - Creates new variable in CDF file.
Result = CDF_VARCREATE(*Id*, *Name* [, *DimVary*] [, /CDF_BYTE | , /CDF_CHAR | , /CDF_DOUBLE | , /CDF_EPOCH | , /CDF_FLOAT | , /CDF_INT1 | , /CDF_INT2 | , /CDF_INT4 | , /CDF_REAL4 | , /CDF_REAL8 | , /CDF_UCHAR | , /CDF_UINT1 | , /CDF_UINT2 | , /CDF_UINT4] [, ALLOCATERECS=*records*] [, DIMENSIONS=*array*] [, NUMELEM=*characters*] [, /REC_NOVARY | , /REC_VARY] [, /ZVARIABLE])

CDF_VARDELETE - Deletes variable from a SINGLE_FILE CDF file.
CDF_VARDELETE, *Id*, *Variable* [, /ZVARIABLE]

CDF_VARGET - Reads multiple values from CDF file variable.
CDF_VARGET, *Id*, *Variable*, *Value* [, COUNT=*vector*] [, INTERVAL=*vector*] [, OFFSET=*vector*] [, REC_COUNT=*records*] [, REC_INTERVAL=*value*] [, REC_START=*record*] [, /STRING{data in CDF file must be type CDF_CHAR or CDF_UCHAR}] [, /ZVARIABLE]

CDF_VARGET1 - Reads one value from a CDF file variable.
CDF_VARGET1, *Id*, *Variable*, *Value* [, OFFSET=*vector*] [, REC_START=*record*] [, /STRING{data in CDF file must be type CDF_CHAR or CDF_UCHAR}] [, /ZVARIABLE]

CDF_VARINQ - Returns structure containing information about specified variable.
Result = CDF_VARINQ(*Id*, *Variable* [, /ZVARIABLE])

CDF_VARNUM - Returns variable number associated with given variable name.
Result = CDF_VARNUM(*Id*, *VarName* [, *IsZVar*])

CDF_VARPUT - Writes value to a variable.
CDF_VARPUT, *Id*, *Variable*, *Value* [, COUNT=*vector*] [, INTERVAL=*vector*] [, OFFSET=*vector*] [, REC_INTERVAL=*value*] [, REC_START=*record*] [, /ZVARIABLE]

CDF_VARRENAME - Renames existing variable.
CDF_VARRENAME, *Id*, *OldVariable*, *NewName* [, /ZVARIABLE]

EOS Routines

EOS_EH_CONVANG - Converts angles between decimal degrees, radians, and packed degrees-minutes-seconds.
Result = EOS_EH_CONVANG(*inAngle*, *code*)

EOS_EH_GETVERSION - Retrieves the HDF-EOS version string of an HDF-EOS file.
Result = EOS_EH_GETVERSION(*fid*, *version*)

EOS_EH_IDINFO - Returns the HDF file IDs corresponding to the HDF-EOS file ID returned by EOS_SW_OPEN, EOS_GD_OPEN, or EOS_PT_OPEN.
Result = EOS_EH_IDINFO(*fid*, *HDFfid*, *sdInterfaceID*)

EOS_EXISTS - Returns True if HDF EOS format library is supported on the current IDL platform.
Result = EOS_EXISTS( )

EOS_GD_ATTACH - Attaches to the grid using the gridname parameter as the identifier.
Result = EOS_GD_ATTACH(*fid*, *gridname*)

EOS_GD_ATTRINFO - Returns number type and number of elements (count) of a grid attribute.
Result = EOS_GD_ATTRINFO(*gridID*, *attrname*, *numbertype*, *count*)

EOS_GD_CLOSE - Closes the HDF grid file.
Result = EOS_GD_CLOSE(*fid*)

EOS_GD_COMPINFO - Returns the compression code and compression parameters for a given field.
Result = EOS_GD_COMPINFO(*gridID*, *fieldname*, *compcode*, *compparm*)

EOS_GD_CREATE - Creates a grid within the file.
Result = EOS_GD_CREATE(*fid*, *gridname*, *xdimsize*, *ydimsize*, *upleftpt*, *lowrightpt*)

EOS_GD_DEFBOXREGION - Defines a longitude-latitude box region for a grid.
Result = EOS_GD_DEFBOXREGION(*gridID*, *cornerlon*, *cornerlat*)

EOS_GD_DEFCOMP - Sets the HDF field compression for subsequent grid field definitions.
Result = EOS_GD_DEFCOMP(*gridID*, *compcode* [, *compparm*])

EOS_GD_DEFDIM - Defines dimensions used by field definition routines to establish size of the field.
Result = EOS_GD_DEFDIM(*gridID*, *dimname*, *dim*)

EOS_GD_DEFFIELD - Defines data fields to be stored in the grid.
Result = EOS_GD_DEFFIELD(*gridID*, *fieldname*, *dimlist*, *numbertype* [, /MERGE])

EOS_GD_DEFORIGIN - Defines the origin of the grid data.
Result = EOS_GD_DEFORIGIN(*gridID*, *origincode*)

EOS_GD_DEFPIXREG - Defines whether the pixel center or pixel corner is used when requesting the location of a given pixel.
Result = EOS_GD_DEFPIXREG(*gridID*, *pixreg*)

EOS_GD_DEFPROJ - Defines the GCTP projection and projection parameters of the grid.
Result = EOS_GD_DEFPROJ(*gridID*, *projcode*, *zonecode*, *spherecode*, *projparm*)

EOS_GD_DEFTILE - Defines the tiling dimensions for fields defined following this function call.
Result = EOS_GD_DEFTILE(*gridID*, *tilecode* [, *tilerank*, *tiledims*])

EOS_GD_DEFTIMEPERIOD - Defines a time period for a grid.
Result = EOS_GD_DEFTIMEPERIOD(*gridID*, *periodID*, *starttime*, *stoptime*)

EOS_GD_DEFVRTREGION - Subsets on a monotonic field or contiguous elements of a dimension.
Result = EOS_GD_DEFVRTREGION(*gridID*, *regionID*, *vertObj*, *range*)

EOS_GD_DETACH - Detaches from grid interface.
Result = EOS_GD_DETACH(*gridID*)

EOS_GD_DIMINFO - Retrieves the size of the specified dimension.
Result = EOS_GD_DIMINFO(*gridID*, *dimname*)

EOS_GD_DUPREGION - Copies information stored in current region or period to a new region or period.
Result = EOS_GD_DUPREGION(*regionID*)

EOS_GD_EXTRACTREGION - Reads data into the data buffer from a subsetted region as defined by EOS_GD_DEFBOXREGION.
Result = EOS_GD_EXTRACTREGION(*gridID*, *regionID*, *fieldname*, *buffer*)

EOS_GD_FIELDINFO - Retrieves information on a specific data field.
Result = EOS_GD_FIELDINFO(*gridID*, *fieldname*, *rank*, *dims*, *numbertype*, *dimlist*)

EOS_GD_GETFILLVALUE - Retrieves fill value for specified field.
Result = EOS_GD_GETFILLVALUE(*gridID*, *fieldname*, *fillvalue*)

EOS_GD_GETPIXELS - Returns the pixel rows and columns for specified longitude/latitude pairs.
Result = EOS_GD_GETPIXELS(*gridID*, *nLonLat*, *lonVal*, *latVal*, *pixRow*, *pixCol*)

EOS_GD_GETPIXVALUES - Reads data from a data field for the specified pixels.
Result = EOS_GD_GETPIXVALUES(*gridID*, *nPixels*, *pixRow*, *pixCol*, *fieldname*, *buffer*)

EOS_GD_GRIDINFO - Returns number of rows, columns, and the location of the upper left and lower right corners of the grid image.
Result = EOS_GD_GRIDINFO(*gridID*, *xdimsize*, *ydimsize*, *upleft*, *lowright*)

EOS_GD_INQATTRS - Retrieves information about attributes defined in grid.
Result = EOS_GD_INQATTRS(*gridID*, *attrlist* [, LENGTH (OUT)=*value*])

EOS_GD_INQDIMS - Retrieves information about dimensions defined in grid.
Result = EOS_GD_INQDIMS(*gridID*, *dimname*, *dims*)

EOS_GD_INQFIELDS - Retrieves information about the data fields defined in grid.
Result = EOS_GD_INQFIELDS(*gridID*, *fieldlist*, *rank*, *numbertype*)

EOS_GD_INQGRID - Retrieves number and names of grids defined in HDF-EOS file.
Result = EOS_GD_INQGRID(*filename*, *gridlist* [, LENGTH (OUT)=*value*])

EOS_GD_INTERPOLATE - Performs bilinear interpolation on a grid field.
Result = EOS_GD_INTERPOLATE(*gridID*, *Interp*, *lonVal*, *latVal*, *fieldname*, *interpVal*)

EOS_GD_NENTRIES - Returns number of entries and descriptive string buffer size for a specified entity.
Result = EOS_GD_NENTRIES(*gridID*, *entrycode* [, LENGTH (OUT)=*value*])

EOS_GD_OPEN - Opens an existing file or creates a new file.
Result = EOS_GD_OPEN(*filename*, *access* [, /CREATE] [, /RDWR | , /READ])

EOS_GD_ORIGININFO - Retrieves origin code.
Result = EOS_GD_ORIGININFO(*gridID*, *origincode*)

EOS_GD_PIXREGINFO - Retrieves the pixel registration code.
Result = EOS_GD_PIXREGINFO(*gridID*, *pixregcode*)

EOS_GD_PROJINFO - Retrieves GCTP projection code, zone code, spheroid code, and projection parameters of the grid.
Result = EOS_GD_PROJINFO(*gridID, projcode, zonecode, spherecode, projparm*)

EOS_GD_QUERY - Returns information about a specified grid.
Result = EOS_GD_QUERY(*Filename, GridName*, [*Info*])

EOS_GD_READATTR - Reads attribute from a grid.
Result = EOS_GD_READATTR(*gridID, attrname, datbuf*)

EOS_GD_READFIELD - Reads data from a grid field.
Result = EOS_GD_READFIELD(*gridID, fieldname, buffer* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_GD_READTILE - Reads from tile within field.
Result = EOS_GD_READTILE(*gridID, fieldname, tilecoords, buffer*)

EOS_GD_REGIONINFO - Returns information about a subsetted region for a particular field.
Result = EOS_GD_REGIONINFO(*gridID, regionID, fieldname, ntype, rank, dims, size, upleftpt, lowrightpt*)

EOS_GD_SETFILLVALUE - Sets fill value for the specified field.
Result = EOS_GD_SETFILLVALUE(*gridID, fieldname, fillvalue*)

EOS_GD_SETTILECACHE - Sets tile cache parameters.
Result = EOS_GD_SETTILECACHE(*gridID, fieldname, maxcache, cachecode*)

EOS_GD_TILEINFO - Returns tiling code, tiling rank, and tiling dimensions for a given field.
Result = EOS_GD_TILEINFO(*gridID, fieldname, tilecode, tilerank, tiledims*)

EOS_GD_WRITEATTR - Writes/updates attribute in a grid.
Result = EOS_GD_WRITEATTR(*gridID, attrname, datbuf* [, COUNT=*value*] [, HDF_TYPE=*value*])

EOS_GD_WRITEFIELD - Writes data to a grid field.
Result = EOS_GD_WRITEFIELD(*gridID, fieldname, data* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_GD_WRITEFIELDMETA - Writes field metadata for a grid field not defined by the Grid API.
Result = EOS_GD_WRITEFIELDMETA(*gridID, fieldname, dimlist, numbertype*)

EOS_GD_WRITETILE - Writes a single tile of data to a field.
Result = EOS_GD_WRITETILE(*gridID, fieldname, tilecoords, data*)

EOS_PT_ATTACH - Attaches to point using the pointname parameter as the identifier.
Result = EOS_PT_ATTACH(*fid, pointname*)

EOS_PT_ATTRINFO - Returns number type and number of elements of a point attribute.
Result = EOS_PT_ATTRINFO(*pointID, attrname, numbertype, count*)

EOS_PT_BCKLINKINFO - Returns linkfield to the previous level.
Result = EOS_PT_BCKLINKINFO(*pointID, level, linkfield*)

EOS_PT_CLOSE - Closes the HDF point file.
Result = EOS_PT_CLOSE(*fid*)

EOS_PT_CREATE - Creates point as a Vgroup within the HDF file.
Result = EOS_PT_CREATE(*fid, pointname*)

EOS_PT_DEFBOXREGION - Defines area of interest for a point.
Result = EOS_PT_DEFBOXREGION(*pointID, cornerlon, cornerlat*)

EOS_PT_DEFLEVEL - Defines a level within a point.
Result = EOS_PT_DEFLEVEL(*pointID, levelname, fieldlist, fieldtype, fieldorder*)

EOS_PT_DEFLINKAGE - Defines linkfield between two levels.
Result = EOS_PT_DEFLINKAGE(*pointID, parent, child, linkfield*)

EOS_PT_DEFTIMEPERIOD - Defines a time period for a point.
Result = EOS_PT_DEFTIMEPERIOD(*pointID, starttime, stoptime*)

EOS_PT_DEFVRTREGION - Selects records within a point whose field values are within a given range.
Result = EOS_PT_DEFVRTREGION(*pointID, regionID, vertObj, range*)

EOS_PT_DETACH - Detaches from a point data set.
Result = EOS_PT_DETACH(*pointID*)

EOS_PT_EXTRACTPERIOD - Reads data from the designated level fields into the data buffer from the subsetted time period.
Result = EOS_PT_EXTRACTPERIOD(*pointID, periodID, level, fieldlist, buffer*)

EOS_PT_EXTRACTREGION - Reads data from the designated level fields into the data buffer from the subsetted area of interest.
Result = EOS_PT_EXTRACTREGION(*pointID, regionID, level, fieldlist, buffer*)

EOS_PT_FWDLINKINFO - Returns linkfield to the given level.
Result = EOS_PT_FWDLINKINFO(*pointID, level, linkfield*)

EOS_PT_GETLEVELNAME - Returns the name of a level given the level number (0-based).
Result = EOS_PT_GETLEVELNAME(*pointID, level, levelname* [, LENGTH (OUT)=*variable*])

EOS_PT_GETRECNUMS - Returns record numbers in one level that are connected to a given set of records in a different level.
Result = EOS_PT_GETRECNUMS(*pointID, inlevel, outlevel, inNrec, inRecs, outNrec, outRecs*)

EOS_PT_INQATTRS - Returns attribute list as a comma-separated string.
Result = EOS_PT_INQATTRS(*pointID, attrlist* [, LENGTH=*value*])

EOS_PT_INQPOINT - Retrieves number and names of points defined in HDF-EOS file.
Result = EOS_PT_INQPOINT*(filename, pointlist* [, LENGTH (OUT)=*value*])

EOS_PT_LEVELINDX - Returns the level index for a given level.
Result = EOS_PT_LEVELINDX(*pointID, levelname*)

EOS_PT_LEVELINFO - Returns information about the fields in a given level.
Result = EOS_PT_LEVELINFO(*pointID, level, fieldlist, fldtype, fldorder*)

EOS_PT_NFIELDS - Returns the number of fields in a level.
Result = EOS_PT_NFIELDS(*pointID, level* [, LENGTH=*bytes*])

EOS_PT_NLEVELS - Returns the number of levels in a point.
Result = EOS_PT_NLEVELS(*pointID*)

EOS_PT_NRECS - Returns the number of records in a given level.
Result = EOS_PT_NRECS(*pointID, level*)

EOS_PT_OPEN - Creates a new file or opens an existing one.
Result = EOS_PT_OPEN(*fieldname* [, /CREATE] [, /RDWR | , /READ])

EOS_PT_PERIODINFO - Returns information about a subsetted time period for a given fieldlist.
Result = EOS_PT_PERIODINFO(*pointID, periodID, level, fieldlist, size*)

EOS_PT_PERIODRECS - Returns record numbers within a subsetted time period for a given level.
Result = EOS_PT_PERIODRECS(*pointID, periodID, level, nrec, recs*)

EOS_PT_QUERY - Returns information about a specified point.
Result = EOS_PT_QUERY(*Filename, PointName*, [*Info*])

EOS_PT_READATTR - Reads attributes.
Result = EOS_PT_READATTR(*pointID, attrname, datbuf*)

EOS_PT_READLEVEL - Reads data from the specified fields and records of a single level in a point.
Result = EOS_PT_READLEVEL(*pointID, level, fieldlist, nrec, recs, buffer*)

EOS_PT_REGIONINFO - Returns information about a subsetted area of interest for a given fieldlist.
Result = EOS_PT_REGIONINFO(*pointID, regionID, level, fieldlist, size*)

EOS_PT_REGIONRECS - Returns the record numbers within a subsetted geographic region for a given level.
Result = EOS_PT_REGIONRECS(*pointID, regionID, level, nrec, recs*)

EOS_PT_SIZEOF - Returns information about specified fields in a point regardless of level.
Result = EOS_PT_SIZEOF(*pointID, fieldlist, fldlevel*)

EOS_PT_UPDATELEVEL - Updates the specified fields and records of a single level.
Result = EOS_PT_UPDATELEVEL(*pointID, level, field, list, nrec, recs, data*)

EOS_PT_WRITEATTR - Writes/updates an attribute in a point.
Result = EOS_PT_WRITEATTR(*pointID, attrname, datbuf* [, COUNT=*value*] [, HDF_TYPE=*value*])

EOS_PT_WRITELEVEL - Writes (appends) full records to a level.
Result = EOS_PT_WRITELEVEL(*pointID, level, nrec, data*)

EOS_QUERY - Returns information about the makeup of an HDF-EOS file.
Result = EOS_QUERY(*Filename*, [*Info*])

EOS_SW_ATTACH - Attaches to the swath using the swathname parameter as the identifier.
Result = EOS_SW_ATTACH(*fid, swathname*)

EOS_SW_ATTRINFO - Returns number type and number of elements of a swath attribute.
Result = EOS_SW_ATTRINFO(*swathID, attrname, numbertype, count*)

EOS_SW_CLOSE - Closes the HDF swath file.
Result = EOS_SW_CLOSE(*fid*)

EOS_SW_COMPINFO - Returns compression code and compression parameters for a given field.
Result = EOS_SW_COMPINFO(*swathID, fieldname, compcode, compparm*)

EOS_SW_CREATE - Creates a swath within the file.
Result = EOS_SW_CREATE(*fid, swathname*)

EOS_SW_DEFBOXREGION - Defines a longitude-latitude box region for a swath.
Result = EOS_SW_DEFBOXREGION(*swathID, cornerlon, cornerlat, mode*)

EOS_SW_DEFCOMP - Sets HDF field compression for subsequent swath field definitions.
Result = EOS_SW_DEFCOMP(*swathID, compcode*, [, *compparm*])

EOS_SW_DEFDATAFIELD - Defines geolocation fields to be stored in the swath.
Result = EOS_SW_DEFDATAFIELD(*swathID, fieldname, dimlist, numbertype* [, /MERGE])

EOS_SW_DEFDIM - Defines dimensions that are used by the field definition routines to establish the size of the field.
Result = EOS_SW_DEFDIM(*swathID, fieldname, dim*)

EOS_SW_DEFDIMMAP - Defines monotonic mapping between the geolocation and data dimensions.
Result = EOS_SW_DEFDIMMAP(*swathID, geodim, datadim, offset, increment*)

EOS_SW_DEFGEOFIELD - Defines geolocation fields to be stored in the swath.
Result = EOS_SW_DEFGEOFIELD(*swathID, fieldname, dimlist, numbertype* [, /MERGE])

EOS_SW_DEFIDXMAP - Defines mapping between a geolocation and data dimension.
Result = EOS_SW_DEFIDXMAP(*swathID, geodim, datadim, index*)

EOS_SW_DEFTIMEPERIOD - Defines a time period for a swath.
Result = EOS_SW_DEFTIMEPERIOD(*swathID, starttime , stoptime, mode*)

EOS_SW_DEFVRTREGION - Subsets along any dimension.
Result = EOS_SW_DEFVRTREGION(*swathID, regionID, vertObj, range*)

EOS_SW_DETACH - Detaches from the swath interface.
Result = EOS_SW_DETACH(*swathID*)

EOS_SW_DIMINFO - Retrieves the size of the specified dimension.
Result = EOS_SW_DIMINFO(*swathID, dimname*)

EOS_SW_DUPREGION - Copies information stored in a current region or period to a new region or period.
Result = EOS_SW_DUPREGION(*regionID*)

EOS_SW_EXTRACTPERIOD - Reads data into the data buffer from the subsetted time period.
Result = EOS_SW_EXTRACTPERIOD(*swathID, periodID, fieldname, external_mode, buffer*)

EOS_SW_EXTRACTREGION - Reads data into the data buffer from the subsetted region.
Result = EOS_SW_EXTRACTREGION(*swathID, regionID, fieldname, external_mode, buffer*)

EOS_SW_FIELDINFO - Retrieves information on a specific data field.
Result = EOS_SW_FIELDINFO(*swathID, fieldname, rank, dims, numbertype, dimlist*)

EOS_SW_GETFILLVALUE - Retrieves fill value for given field.
Result = EOS_SW_GETFILLVALUE(*swathID, fieldname, fillvalue*)

EOS_SW_IDXMAPINFO - Retrieves size of the indexed array and the array of indexed elements of the specified geolocation mapping.
Result = EOS_SW_IDXMAPINFO(*swathID, geodim, datadim, index*)

EOS_SW_INQATTRS - Retrieves information about attributes defined in swath.
Result = EOS_SW_INQATTRS(*swathID, attrlist* [, LENGTH (OUT)=*value*])

EOS_SW_INQDATAFIELDS - Retrieves information about all of the data fields defined in swath.
Result = EOS_SW_INQDATAFIELDS(*swathID, fieldlist, rank, numbertype*)

EOS_SW_INQDIMS - Retrieves information about all of the dimensions defined in swath.
Result = EOS_SW_INQDIMS(*swathID, dimname, dim*)

EOS_SW_INQGEOFIELDS - Retrieves information about all of the geolocation fields defined in swath.
Result = EOS_SW_INQGEOFIELDS(*swathID, fieldlist, rank, numbertype*)

EOS_SW_INQIDXMAPS - Retrieves information about all indexed geolocation/data mappings in swath.
Result = EOS_SW_INQIDXMAPS(*swathID, idxmap, idxsizes*)

EOS_SW_INQMAPS - Retrieves information about all non-indexed geolocation relations in swath.
Result = EOS_SW_INQMAPS(*swathID, dimmap, offset, increment*)

EOS_SW_INQSWATH - Retrieves number and names of swaths defined in HDF-EOS file.
Result = EOS_SW_INQSWATH(*filename, swathlist* [, LENGTH=*value*])

EOS_SW_MAPINFO - Retrieves offset and increment of the specified geolocation mapping.
Result = EOS_SW_MAPINFO(*swathID, geodim, datadim, offset, increment*)

EOS_SW_NENTRIES - Returns number of entries and descriptive string buffer size for specified entity.
Result = EOS_SW_NENTRIES(*swathID, entrycode* [, LENGTH (OUT)=*value*])

EOS_SW_OPEN - Opens an existing file, or creates a new file.
Result = EOS_SW_OPEN(*filename* [, /CREATE] [, /RDWR | , /READ])

EOS_SW_PERIODINFO - Returns information about a subsetted time period for a given field.
Result = EOS_SW_PERIODINFO(*swathID, periodID, fieldname, ntype, rank, dims, size*)

EOS_SW_QUERY - Returns information about a specified swath.
Result=EOS_SW_QUERY(*Filename, SwathName,* [*Info*])

EOS_SW_READATTR - Reads attribute from a swath field.
Result = EOS_SW_READATTR(*swathID, attrname, datbuf*)

EOS_SW_READFIELD - Reads data from a swath field.
Result = EOS_SW_READFIELD(*swathID, fieldname, buffer* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_SW_REGIONINFO - Returns information about a subsetted region for a given field.
Result = EOS_SW_REGIONINFO(*swathID, regionID, fieldname, ntype, rank, dims, size*)

EOS_SW_SETFILLVALUE - Sets fill value for the specified field.
Result = EOS_SW_SETFILLVALUE(*swathID, fieldname, fillvalue*)

EOS_SW_WRITEATTR - Writes/updates attribute in a swath.
Result = EOS_SW_WRITEATTR(*swathID*, *attrname*, *datbuf* [, COUNT=*value*] [, HDF_TYPE=*value*])

EOS_SW_WRITEDATAMETA - Writes field metadata for an existing data field.
Result = EOS_SW_WRITEDATAMETA(*swathID*, *fieldname*, *dimlist*, *numbertype*)

EOS_SW_WRITEFIELD - Writes data to a swath field.
Result = EOS_SW_WRITEFIELD(*swathID*, *fieldname*, *cut*, *data* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_SW_WRITEGEOMETA - Writes field metadata for an existing geolocation field.
Result = EOS_SW_WRITEGEOMETA(*swathID*, *fieldname*, *dimlist*, *numbertype*)

HDF Routines

HDF_AN_ANNLEN - Returns number of characters in annotation.
Result = HDF_AN_ANNLEN(*ann_id*)

HDF_AN_ANNLIST - Obtains a list of annotation identifiers.
Result = HDF_AN_ANNLIST(*an_id*, *annot_type*, *obj_tag*, *obj_ref*, *ann_list*)

HDF_AN_ATYPE2TAG - Returns HDF tag corresponding to given annotation type.
Result = HDF_AN_ATYPE2TAG(*annot_type*)

HDF_AN_CREATE - Creates HDF AN annotation.
Result = HDF_AN_CREATE(*an_id*, *obj_tag*, *obj_ref*, *annot_type*)

HDF_AN_CREATEF - Creates file annotation.
Result = HDF_AN_CREATEF(*an_id*, *annot_type*)

HDF_AN_END - Terminates access to the HDF AN interface.
HDF_AN_END, *an_id*

HDF_AN_ENDACCESS - Terminates access to an annotation.
HDF_AN_ENDACCESS, *ann_id*

HDF_AN_FILEINFO - Retrieves total number of annotations and stores them in the appropriate parameters.
Result = HDF_AN_FILEINFO(*an_id*, *n_file_labels*, *n_file_descs*, *n_data_labels*, *n_data_descs*)

HDF_AN_GET_TAGREF - Retrieves HDF tag and reference number of annotation.
Result = HDF_AN_GET_TAGREF(*an_id*, *index*, *annot_type*, *ann_tag*, *ann_ref*)

HDF_AN_ID2TAGREF - Retrieves HDF tag/reference number pair of annotation.
Result = HDF_AN_ID2TAGREF(*ann_id*, *ann_tag*, *ann_ref*)

HDF_AN_NUMANN - Returns total number of annotations of a given type.
Result = HDF_AN_NUMANN(*an_id*, *annot_type*, *obj_tag*, *obj_ref*)

HDF_AN_READANN - Reads specified annotation.
Result = HDF_AN_READANN(*ann_id*, *annotation* [, LENGTH=*characters*])

HDF_AN_SELECT - Obtains identifier of specified annotation.
Result = HDF_AN_SELECT(*an_id*, *index*, *annot_type*)

HDF_AN_START - Initializes interface for specified file.
Result = HDF_AN_START(*file_id*)

HDF_AN_TAG2ATYPE - Returns annotation type of corresponding HDF tag.
Result = HDF_AN_TAG2ATYPE(*ann_tag*)

HDF_AN_TAGREF2ID - Returns ID of annotation with given tag.
Result = HDF_AN_TAGREF2ID(*an_id*, *ann_tag*, *ann_ref*)

HDF_AN_WRITEANN - Writes annotation text.
Result = HDF_AN_WRITEANN(*ann_id*, *annotation* [, LENGTH=*characters*])

HDF_BROWSER - See "HDF_BROWSER" on page 44.

HDF_CLOSE - Closes HDF file associated with the given file handle.
HDF_CLOSE, *FileHandle*

HDF_DELDD - Deletes tag or reference from list of data descriptors.
HDF_DELDD, *FileHandle*, *Tag*, *Ref*

HDF_DF24_ADDIMAGE - Writes 24-bit raster image to HDF file.
HDF_DF24_ADDIMAGE, *Filename*, *Image* [, /FORCE_BASELINE{useful only if QUALITY<25}] [, /JPEG | , /RLE] [, QUALITY=*value*{0 to 100}]

HDF_DF24_GETIMAGE - Reads 24-bit raster image from HDF file.
HDF_DF24_GETIMAGE, *Filename*, *Image* [, /LINE | , /PIXEL | , /PLANE]

HDF_DF24_GETINFO - Retrieves information about the current 24-bit HDF image.
HDF_DF24_GETINFO, *Filename*, *Width*, *Height*, *Interlace*

HDF_DF24_LASTREF - Returns reference number of most recently read or written 24-bit image in an HDF file.
Result = HDF_DF24_LASTREF( )

HDF_DF24_NIMAGES - Returns the number of 24-bit images in an HDF file.
Result = HDF_DF24_NIMAGES(*Filename*)

HDF_DF24_READREF - Sets reference number of image in an HDF file.
HDF_DF24_READREF, *Filename*, *Refno*

HDF_DF24_RESTART - Causes next call to HDF_DF24_GETIMAGE to read first 24-bit image in the HDF file.

HDF_DF24_RESTART

HDF_DFAN_ADDFDS - Adds file description to HDF file.

HDF_DFAN_ADDFDS, *Filename*, *Description*

HDF_DFAN_ADDFID - Adds file annotation to HDF file.

HDF_DFAN_ADDFID, *Filename*, *Label*

HDF_DFAN_GETDESC - Reads description for given tag and reference number in HDF file.

HDF_DFAN_GETDESC, *Filename*, *Tag*, *Ref*, *Description* [, /STRING]

HDF_DFAN_GETFDS - Reads next available file description.

HDF_DFAN_GETFDS, *Filename*, *Description* [, /FIRST] [, /STRING]

HDF_DFAN_GETFID - Reads next available file annotation.

HDF_DFAN_GETFID, *Filename*, *Label* [, /FIRST]

HDF_DFAN_GETLABEL - Reads label for given tag-reference pair.

HDF_DFAN_GETLABEL, *Filename*, *Tag*, *Ref*, *Label*

HDF_DFAN_LABLIST - Retrieves list of reference numbers and labels for given tag.

Result = HDF_DFAN_LABLIST(*Filename*, *Tag*, *Reflist*, *Labellist* [, LISTSIZE=*value*] [, MAXLABEL=*value*] [, STARTPOS=*value*] [, /STRING])

HDF_DFAN_LASTREF - Returns reference number of most recently read or written annotation.

Result = HDF_DFAN_LASTREF( )

HDF_DFAN_PUTDESC - Writes description for given tag and reference number.

HDF_DFAN_PUTDESC, *Filename*, *Tag*, *Ref*, *Description*

HDF_DFAN_PUTLABEL - Writes label for given tag and reference number.

HDF_DFAN_PUTLABEL, *Filename*, *Tag*, *Ref*, *Label*

HDF_DFP_ADDPAL - Appends palette to a HDF file.

HDF_DFP_ADDPAL, *Filename*, *Palette*

HDF_DFP_GETPAL - Reads next available palette from HDF file.

HDF_DFP_GETPAL, *Filename*, *Palette*

HDF_DFP_LASTREF - Returns reference number of most recently read or written palette in HDF file.

Result = HDF_DFP_LASTREF( )

HDF_DFP_NPALS - Returns number of palettes present in HDF file.

Result = HDF_DFP_NPALS(*Filename*)

HDF_DFP_PUTPAL - Appends palette to a HDF file.

HDF_DFP_PUTPAL, *Filename*, *Palette* [, /DELETE] [, /OVERWRITE]

HDF_DFP_READREF - Sets reference number of the palette.

HDF_DFP_READREF, *Filename*, *Refno*

HDF_DFP_RESTART - Causes next call to HDF_DFR8_GETPAL to read from the first palette in HDF file.

HDF_DFP_RESTART

HDF_DFP_WRITEREF - Sets reference number for next palette to be written to a HDF file.

HDF_DFP_WRITEREF, *Filename*, *Refno*

HDF_DFR8_ADDIMAGE - Appends 8-bit raster image to the specified HDF file.

HDF_DFR8_ADDIMAGE, *Filename*, *Image* [, /FORCE_BASELINE{useful only if QUALITY<25}] [, /JPEG | , /RLE] [[, /IMCOMP] , PALETTE=*vector or array*] [, QUALITY=*value*]

HDF_DFR8_GETIMAGE - Retrieves image, palette from HDF file.

HDF_DFR8_GETIMAGE, *Filename*, *Image* [, *Palette*]

HDF_DFR8_GETINFO - Retrieves information about the current 8-bit HDF image.

HDF_DFR8_GETINFO, *Filename*, *Width*, *Height*, *Has_Palette*

HDF_DFR8_LASTREF - Returns reference number of the most recently read or written 8-bit image in HDF file.

Result = HDF_DFR8_LASTREF( )

HDF_DFR8_NIMAGES - Returns number of 8-bit images in specified HDF file.

Result = HDF_DFR8_NIMAGES(*Filename*)

HDF_DFR8_PUTIMAGE - Writes 8-bit raster image as first image in HDF file.

HDF_DFR8_PUTIMAGE, *Filename*, *Image* [, /FORCE_BASELINE{useful only if QUALITY<25}] [[, /IMCOMP] , PALETTE=*vector or array*] [, /JPEG | , /RLE] [, QUALITY=*value*]

HDF_DFR8_READREF - Sets reference number of image to be read from a HDF file by the next call to HDF_DFR8_GETIMAGE.

HDF_DFR8_READREF, *Filename*, *Refno*

HDF_DFR8_RESTART - Causes next call to HDF_DFR8_GETIMAGE to read from first image in HDF file.

HDF_DFR8_RESTART

HDF_DFR8_SETPALETTE - Sets current palette to be used for subsequent images in a HDF file.

HDF_DFR8_SETPALETTE, *Palette*

HDF_DUPDD - Generates new references to existing data in HDF file.

HDF_DUPDD, *FileHandle*, *NewTag*, *NewRef*, *OldTag*, *OldRef*

HDF_EXISTS - Returns True if HDF format library is supported on the current IDL platform.

Result = HDF_EXISTS( )

HDF_GR_ATTRINFO - Retrieves information about specified HDF data object.

Result = HDF_GR_ATTRINFO(*obj_id*, *attr_index*, *name*, *data_type*, *count*)

HDF_GR_CREATE - Creates HDF GR raster image.
Result = HDF_GR_CREATE(*gr_id*, *name*, *ncomp*, *data_type*, *interlace_mode*, *dim_sizes*)

HDF_GR_END - Terminates specified HDF GR interface session.
HDF_GR_END, *gr_id*

HDF_GR_ENDACCESS - Terminates access to specified raster image.
HDF_GR_ENDACCESS, *ri_id*

HDF_GR_FILEINFO - Retrieves number of raster images and global attributes for the specified HDF GR interface.
Result = HDF_GR_FILEINFO(*gr_id*, *n_images*, *n_file_attrs*)

HDF_GR_FINDATTR - Finds index of HDF data object's attribute given its attribute name.
Result = HDF_GR_FINDATTR(*obj_id*, *attr_name*)

HDF_GR_GETATTR - Obtains all values of HDF GR attribute.
Result = HDF_GR_GETATTR(*obj_id*, *attr_index*, *values*)

HDF_GR_GETCHUNKINFO - Retrieves chunking information about HDF GR raster image.
Result = HDF_GR_GETCHUNKINFO(*ri_id*, *dim_length*, *flag*)

HDF_GR_GETIMINFO - Retrieves general information about HDF GR raster image.
Result = HDF_GR_GETIMINFO(*ri_id*, *gr_name*, *ncomp*, *data_type*, *interlace_mode*, *dim_sizes*, *num_attrs*)

HDF_GR_GETLUTID - Gets identifier of HDF GR palette.
Result = HDF_GR_GETLUTID(*ri_id*, *pal_index*)

HDF_GR_GETLUTINFO - Retrieves information about a palette.
Result = HDF_GR_GETLUTINFO(*pal_id*, *ncomp*, *data_type*, *interlace_mode*, *num_entries*)

HDF_GR_IDTOREF - Returns HDF reference number of specified raster image.
Result = HDF_GR_IDTOREF(*ri_id*)

HDF_GR_LUTTOREF - Returns HDF reference number of the specified palette.
Result = HDF_GR_LUTTOREF(*pal_id*)

HDF_GR_NAMETOINDEX - Returns index of raster image given its name
Result = HDF_GR_NAMETOINDEX(*gr_id*, *gr_name*)

HDF_GR_READIMAGE - Reads subsample of raster image.
Result = HDF_GR_READIMAGE(*ri_id*, *data*
[, EDGE=*array*] [, /INTERLACE] [, START=*array*]
[, STRIDE=*array*])

HDF_GR_READLUT - Reads specified palette.
Result = HDF_GR_READLUT(*pal_id*, *pal_data*
[, /INTERLACE])

HDF_GR_REFTOINDEX - Returns index of specified raster image.
Result = HDF_GR_REFTOINDEX(*gr_id*, *gr_ref*)

HDF_GR_SELECT - Obtains identifier of specified raster image.
Result = HDF_GR_SELECT(*gr_id*, *index*)

HDF_GR_SETATTR - Attaches attribute to specified object.
Result = HDF_GR_SETATTR(*obj_id*, *attr_name*, *data_type*, *count*, *values*)

HDF_GR_SETCHUNK - Makes specified raster image a chunked raster image.
Result = HDF_GR_SETCHUNK(*ri_id*, *dim_length*, *comp_type*, *comp_prm*)

HDF_GR_SETCHUNKCACHE - Sets maximum number of chunks to be cached.
Result = HDF_GR_SETCHUNKCACHE(*ri_id*, *maxcache*, *flags*)

HDF_GR_SETCOMPRESS - Specifies whether specified raster image will be stored in compressed format.
Result = HDF_GR_SETCOMPRESS(*ri_id*, *comp_type*, *comp_prm*)

HDF_GR_SETEXTERNALFILE - Specifies that raster image will be written to external file.
Result = HDF_GR_SETEXTERNALFILE(*ri_id*, *filename*, *offset*)

HDF_GR_START - Initializes interface for the specified file.
Result = HDF_GR_START(*file_id*)

HDF_GR_WRITEIMAGE - Writes subsample of raster image data.
Result = HDF_GR_WRITEIMAGE(*ri_id*, *data*
[, EDGE=*array*] [, INTERLACE={0 | 1 | 2}]
[, START=*array*] [, STRIDE=*array*])

HDF_GR_WRITELUT - Writes a palette.
Result = HDF_GR_WRITELUT(*pal_id*, *pal_data*)

HDF_HDF2IDLTYPE - Converts HDF data type code into IDL variable type code.
Result = HDF_HDF2IDLTYPE(*hdftypecode*)

HDF_IDL2HDFTYPE - Converts IDL variable type code into HDF data type code.
Result = HDF_IDL2HDFTYPE(*idltypecode*)

HDF_ISHDF - Determines whether specified file is HDF file.
Result = HDF_ISHDF(*Filename*)

HDF_LIB_INFO - Returns information about the HDF Library being used.
HDF_LIB_INFO, [*FileHandle*] [, MAJOR=*variable*]
[, MINOR=*variable*] [, RELEASE=*variable*]
[, VERSION=*variable*]

HDF_NEWREF - Returns next available reference number for HDF file.
Result = HDF_NEWREF(*FileHandle*)

HDF_NUMBER - Returns number of tags in HDF file or the number of references associated with a given tag.
Result = HDF_NUMBER(*FileHandle* [, TAG=*integer*])

HDF_OPEN - Opens or creates HDF file for reading and/or writing.

Result = HDF_OPEN(*Filename* [, /ALL] [, /CREATE] [, NUM_DD=*value*] [, /RDWR] [, /READ] [, /WRITE])

HDF_PACKDATA - Packs a set IDL variable into an array of raw byte data.

Result = HDF_PACKDATA(*data1* [, *data2* [, *data3* [, *data4* [, *data5* [, *data6* [, *data7* [, *data8*]]]]]]] [, HDF_ORDER=*array*] [, HDF_TYPE=*array*] [, NREC=*records*])

HDF_READ - See "HDF_READ" on page 44.

HDF_SD_ADDDATA - Writes hyperslab of values to an SD dataset.

HDF_SD_ADDDATA, *SDS_ID*, *Data* [, COUNT=*vector*] [, /NOREVERSE] [, START=*vector*] [, STRIDE=*vector*]

HDF_SD_ATTRFIND - Locates index of HDF attribute given its name.

Result = HDF_SD_ATTRFIND(*S_ID*, *Name*)

HDF_SD_ATTRINFO - Reads or retrieves information about SD attribute.

HDF_SD_ATTRINFO, *S_ID*, *Attr_Index* [, COUNT=*variable*] [, DATA=*variable*] [, HDF_TYPE=*variable*] [, NAME=*variable*] [, TYPE=*variable*]

HDF_SD_ATTRSET - Writes attributes to an open HDF SD dataset.

HDF_SD_ATTRSET, *S_ID*, *Attr_Name*, *Values* [, *Count*] [, /BYTE] [, /DFNT_CHAR] [, /DFNT_FLOAT32] [, /DFNT_FLOAT64] [, /DFNT_INT8] [, /DFNT_INT16] [, /DFNT_INT32] [, /DFNT_UINT8] [, /DFNT_UINT16] [, /DFNT_UINT32] [, /DOUBLE] [, /FLOAT] [, /INT] [, /LONG] [, /SHORT] [, /STRING]

HDF_SD_CREATE - Creates and defines a Scientific Dataset for an HDF file.

Result = HDF_SD_CREATE(*SD_ID*, *Name*, *Dims* [, /BYTE] [, /DFNT_CHAR8] [, /DFNT_FLOAT32] [, /DFNT_FLOAT64] [, /DFNT_INT8] [, /DFNT_INT16] [, /DFNT_INT32] [, /DFNT_UINT8] [, /DFNT_UINT16] [, /DFNT_UINT32] [, /DOUBLE] [, /FLOAT] [, HDF_TYPE=*type*] [, /INT] [, /LONG] [, /SHORT] [, /STRING])

HDF_SD_DIMGET - Retrieves info. about SD dataset dimension.

HDF_SD_DIMGET, *Dim_ID* [, /COUNT] [, COMPATIBILITY=*variable*] [, /FORMAT] [, /LABEL] [, /NAME] [, /NATTR] [, /SCALE] [, /TYPE] [, /UNIT]

HDF_SD_DIMGETID - Returns dimension ID given a dataset "SDS_ID" and dimension number.

Result = HDF_SD_DIMGETID(*SDS_ID*, *Dimension_Number*)

HDF_SD_DIMSET - Sets scale and data strings for SD dimension.

HDF_SD_DIMSET, *Dim_ID* [, /BW_INCOMP] [, FORMAT=*string*] [, LABEL=*string*] [, NAME=*string*] [, SCALE=*vector*] [, UNIT=*string*]

HDF_SD_END - Closes SD interface to an HDF file.

HDF_SD_END, *SD_ID*

HDF_SD_ENDACCESS - Closes SD dataset interface.

HDF_SD_ENDACCESS, *SD_ID*

HDF_SD_FILEINFO - Retrieves the number of datasets and global attributes in HDF file.

HDF_SD_FILEINFO, *SD_ID*, *Datasets*, *Attributes*

HDF_SD_GETDATA - Retrieves a hyperslab of values from SD dataset.

HDF_SD_GETDATA, *SDS_ID*, *Data* [, COUNT=*vector*] [, /NOREVERSE] [, START=*vector*] [, STRIDE=*vector*]

HDF_SD_GETINFO - Retrieves information about SD dataset.

HDF_SD_GETINFO, *SDS_ID* [, CALDATA=*variable*] [, COORDSYS=*variable*] [, DIMS=*variable*] [, FILL=*variable*] [, FORMAT=*variable*] [, HDF_TYPE=*variable*] [, LABEL=*variable*] [, NAME=*variable]* [, NATTS=*variable*] [, NDIMS=*variable*] [, /NOREVERSE] [, RANGE=*variable*] [, TYPE=*variable*] [, UNIT=*variable*]

HDF_SD_IDTOREF - Converts SD data set ID into SD data set reference number.

Result = HDF_SD_IDTOREF(*SDS_ID*)

HDF_SD_ISCOORDVAR - Determines whether supplied dataset ID represents NetCDF "coordinate" variable.

Result = HDF_SD_ISCOORDVAR(*SDS_ID*)

HDF_SD_NAMETOINDEX - Returns SD dataset index given its name and SD interface ID.

Result = HDF_SD_NAMETOINDEX(*SD_ID*, *SDS_Name*)

HDF_SD_REFTOINDEX - Returns SD dataset index given its reference number and SD interface ID.

Result = HDF_SD_REFTOINDEX(*SD_ID*, *Refno*)

HDF_SD_SELECT - Returns SD dataset ID.

Result = HDF_SD_SELECT(*SD_ID*, *Number*)

HDF_SD_SETCOMPRESS - Compresses an existing HDF SD dataset or sets the compression method of a new HDF SD dataset.

HDF_SD_SETCOMPRESS, *SDS_ID*, *comptype* [, EFFORT=*integer*{1 to 9}]

HDF_SD_SETEXTFILE - Moves data values from a dataset into an external file.

HDF_SD_SETEXTFILE, *SDS_ID*, *Filename* [, OFFSET=*bytes*]

HDF_SD_SETINFO - Sets information about SD dataset.

HDF_SD_SETINFO, *SDS_ID* [, FILL=*value*] [, FORMAT=*string*] [, LABEL=*string*] [, RANGE=*[max, min]*] [, UNIT=*string*] [, COORDSYS=*string*] [, CALDATA=*structure*]

HDF_SD_START - Opens or creates HDF file and initializes SD interface.
Result = HDF_SD_START(*Filename* [, /READ | , /RDWR] [, /CREATE])

HDF_UNPACKDATA - Unpacks array of byte data into IDL variables.
HDF_UNPACKDATA, *packeddata*, *data1* [, *data2* [, *data3* [, *data4* [, *data5* [, *data6* [, *data7* [, *data8*]]]]]]] [, HDF_ORDER=*array*] [, HDF_TYPE=*array*] [, NREC=*records*]

HDF_VD_ATTACH - Accesses a VData with the given ID.
Result = HDF_VD_ATTACH(*FileHandle*, *VDataId* [, /READ] [, /WRITE])

HDF_VD_ATTRFIND - Returns an attribute's index number given the name of an attribute.
Result = HDF_VD_ATTRFIND(*VData*, *FieldID*, *Name*)

HDF_VD_ATTRINFO - Retrieves information about a VData attribute.
HDF_VD_ATTRINFO, *VData*, *FieldID*, *AttrID*, *Values* [, COUNT=*variable*] [, DATA=*variable*] [, HDF_TYPE=*variable*] [, NAME=*variable*] [, TYPE=*variable*]

HDF_VD_ATTRSET - Writes a vdata attribute or a vdata field attribute to the currently attached HDF VData structure.
HDF_VD_ATTRSET, *VData*, *FieldID*, *Attr_Name*, *Values* [, *Count*] [, /BYTE] [, /DFNT_CHAR8] [, /DFNT_FLOAT32] [, /DFNT_FLOAT64] [, /DFNT_INT8] [, /DFNT_INT16] [, /DFNT_INT32] [, /DFNT_UCHAR8] [, /DFNT_UINT8] [, /DFNT_UINT16] [, /DFNT_UINT32] [, /DOUBLE] [, /FLOAT] [, /INT] [, /LONG] [, /SHORT] [, /STRING] [, /UINT] [, /ULONG]

HDF_VD_DETACH - Called when done accessing a VData.
HDF_VD_DETACH, *VData*

HDF_VD_FDEFINE - Adds new field specification for VData.
HDF_VD_FDEFINE, *VData*, *Fieldname* [, /BYTE | , /DLONG, | , /DOUBLE | , /DULONG | , /FLOAT | , /INT | , /LONG | , /UINT, | , /ULONG] [, ORDER=*value*]

HDF_VD_FEXIST - Returns true if specified fields exist in HDF file.
Result = HDF_VD_FEXIST(*VData*, *Fieldnames*)

HDF_VD_FIND - Returns reference number of specified VData.
Result = HDF_VD_FIND(*FileHandle*, *Name*)

HDF_VD_GET - Returns information about a VData.
HDF_VD_GET, *VData* [, CLASS=*variable*] [, COUNT=*variable*] [, FIELDS=*variable*] [, INTERLACE=*variable*] [, NAME=*variable*] [, NFIELDS=*variable*] [, REF=*variable*] [, SIZE=*variable*] [, TAG=*variable*]

HDF_VD_GETID - Returns VData reference number for next VData.
Result = HDF_VD_GETID(*FileHandle*, *VDataId*)

HDF_VD_GETINFO - Returns information about each Vdata field.
HDF_VD_GETINFO, *VData*, *Index* [, NAME=*variable*] [, ORDER=*variable*] [, SIZE=*variable*] [, TYPE=*variable*]

HDF_VD_INSERT - Adds VData or VGroup to contents of VGroup.
HDF_VD_INSERT, *VGroup*, *VData(or Vgroup)*[, POSITION=*variable*]

HDF_VD_ISATTR - Returns True (1) if the VData is storing an attribute, False (0) otherwise.
Result = HDF_VD_ISATTR(*VData*)

HDF_VD_ISVD - Returns True (1) if an object is a VData.
Result = HDF_VD_ISVD(*VGroup*, *Id*)

HDF_VD_ISVG - Returns True (1) if object is a VGroup.
Result = HDF_VG_ISVG(*VGroup*, *Id*)

HDF_VD_LONE - Returns array containing all VDatas that are not contained in another VData.
Result = HDF_VD_LONE(*FileHandle* [, MAXSIZE=*value*])

HDF_VD_NATTRS - Returns the number of attributes associated with the specified VData.
Result = HDF_ VD _NATTRS(*VData*, *FieldID*)

HDF_VD_READ - Reads data from a VData.
Result = HDF_VD_READ(*VData*, *Data* [, FIELDS=*string*] [, /FULL_INTERLACE | , /NO_INTERLACE] [, NRECORDS=*records*])

HDF_VD_SEEK - Moves read pointer in specified VData to specific record number.
HDF_VD_SEEK, *VData*, *Record*

HDF_VD_SETINFO - Specifies general information about a VData.
HDF_VD_SETINFO, *VData* [, CLASS=*string*] [, /FULL_INTERLACE | , /NO_INTERLACE] [, NAME=*string*]

HDF_VD_WRITE - Stores data in a VData
HDF_VD_WRITE, *VData*, *Fields*, *Data* [, /FULL_INTERLACE | , /NO_INTERLACE] [, NRECORDS=*records*]

HDF_VG_ADDTR - Adds tag and reference to specified VGroup.
HDF_VG_ADDTR, *VGroup*, *Tag*, *Ref*

HDF_VG_ATTACH - Attaches (opens) a VGroup.
Result = HDF_VG_ATTACH(*FileHandle*, *VGroupId* [, /READ] [, /WRITE])

HDF_VG_DETACH - Called when finished accessing a VGroup.
HDF_VG_DETACH, *VGroup*

HDF_VG_GETID - Returns VGroup ID for specified VGroup.
Result = HDF_VG_GETID(*FileHandle*, *VGroupId*)

HDF_VG_GETINFO - Returns information about a VGroup.
HDF_VG_GETINFO, *VGroup* [, CLASS=*variable*] [, NAME=*variable*] [, NENTRIES=*variable*] [, REF=*variable*] [, TAG=*variable*]

HDF_VG_GETNEXT - Returns reference number of the next object in a VGroup.
Result = HDF_VG_GETNEXT(*VGroup*, *Id*)

HDF_VG_GETTR - Returns tag/reference pair at specified position within a VGroup.
HDF_VG_GETTR, *VGroup*, *Index*, *Tags*, *Refs*

HDF_VG_GETTRS - Returns tag/reference pairs of HDF file objects belonging to the specified VGroup.
HDF_VG_GETTRS, *VGroup*, *Tags*, *Refs* [, MAXSIZE=*value*]

HDF_VG_INQTR - Returns true if specified tag/reference pair is linked to the specified Vgroup.
Result = HDF_VG_INQTR(*VGroup*, *Tag*, *Ref*)

HDF_VG_INSERT - Adds VData or VGroup to contents of VGroup.
HDF_VG_INSERT, *VGroup*, *VData(or Vgroup)*[, POSITION=*variable*]

HDF_VG_ISVD - Returns true if object is a VData.
Result = HDF_VG_ISVD(*VGroup*, *Id*)

HDF_VG_ISVG - Returns true if object is a VGroup.
Result = HDF_VG_ISVG(*VGroup*, *Id*)

HDF_VG_LONE - Returns array containing IDs of all VGroups that are not contained in another VGroup.
Result = HDF_VG_LONE(*FileHandle* [, MAXSIZE=*value*])

HDF_VG_NUMBER - Returns number of HDF file objects in specified VGroup.
Result = HDF_VG_NUMBER(*VGroup*)

HDF_VG_SETINFO - Sets the name and class of a VGroup.
HDF_VG_SETINFO, *VGroup* [, CLASSNAME=*string*] [, NAME=*string*]

HDF5 Routines

H5_BROWSER - Presents a graphical user interface for viewing and reading HDF5 files.
Result = H5_BROWSER([*Files*] [, /DIALOG_READ])

H5_CLOSE - Flushes all data to disk, closes file identifiers, and cleans up memory.
H5_CLOSE

H5_CREATE - Creates and closes a new HDF5 file.
H5_CREATE, *Filename*, *Structure*

H5_GET_LIBVERSION - Returns the current version of the HDF5 library used by IDL.
Result = H5_GET_LIBVERSION( )

H5_OPEN - Initializes IDL's HDF5 library.
H5_OPEN

H5_PARSE - Recursively descends through an HDF5 file or group and creates an IDL structure containing object information and data.
Result = H5_PARSE (*File* [, /READ_DATA])
or
Result = H5_PARSE (*Loc_id*, *Name* [, FILE=*string*] [, PATH=*string*] [, /READ_DATA])

H5A_CLOSE - Closes the specified attribute and releases resources used by it.
H5A_CLOSE, *Attribute_id*

H5A_CREATE - Creates a dataset as an attribute of another group or dataset.
Result = H5A_CREATE(*Loc_id*, *Name*, *Datatype_id*, *Dataspace_id*)

H5A_DELETE - Removes the attribute specified by its name from a dataset, group, or named datatype.
H5A_DELETE, *Loc_id*, *Name*

H5A_GET_NAME - Retrieves an attribute name given the attribute identifier number.
Result = H5A_GET_NAME(*Attribute_id*)

H5A_GET_NUM_ATTRS - Returns the number of attributes attached to a group, dataset, or a named datatype.
Result = H5A_GET_NUM_ATTRS(*Loc_id*)

H5A_GET_SPACE - Returns the identifier number of a copy of the dataspace for an attribute.
Result = H5A_GET_SPACE(*Attribute_id*)

H5A_GET_TYPE - Returns the identifier number of a copy of the datatype for an attribute.
Result = H5A_GET_TYPE(*Attribute_id*)

H5A_OPEN_IDX - Opens an existing attribute by the index of that attribute.
Result = H5A_OPEN_IDX(*Loc_id*, *Index*)

H5A_OPEN_NAME - Opens an existing attribute by the name of that attribute.
Result = H5A_OPEN_NAME(*Loc_id*, *Name*)

H5A_READ - Reads the data within an attribute, converting from the HDF5 file datatype into the HDF5 memory datatype, and finally into the corresponding IDL datatype.
Result = H5A_READ(*Attribute_id*)

H5A_WRITE - Writes data to an attribute.
H5A_WRITE, *Attribute_id*, *Data*

H5D_CLOSE - Closes the specified dataset and releases its used resources.
H5D_CLOSE, *Dataset_id*

H5D_CREATE - Creates a dataset at the specified location.
Result = H5D_CREATE(*Loc_id*, *Name*, *Datatype_id*, *Dataspace_id* [, CHUNK_DIMENSIONS=*vector* [, GZIP=*value* [, /SHUFFLE]]])

H5D_EXTEND - Changes the current dimensions of the Dataset, within the limits of the Dataspace.

H5D_EXTEND,*Dataset_id*, *Size*

H5D_GET_SPACE - Returns an identifier number for a copy of the dataspace for a dataset.

Result = H5D_GET_SPACE(*Dataset_id*)

H5D_GET_STORAGE_SIZE - Returns the amount of storage in bytes required for a dataset.

Result = H5D_GET_STORAGE_SIZE(*Dataset_id*)

H5D_GET_TYPE - Returns an identifier number for a copy of the datatype for a dataset.

Result = H5D_GET_TYPE(*Dataset_id*)

H5D_OPEN - Opens an existing dataset within an HDF5 file.

Result = H5D_OPEN(*Loc_id*, *Name*)

H5D_READ - Reads the data within a dataset, converting from the HDF5 file datatype into the HDF5 memory datatype, and finally into the corresponding IDL datatype.

Result = H5D_READ(*Dataset_id* [, FILE_SPACE=*id*] [, MEMORY_SPACE=*id*])

H5D_WRITE - Writes data to a dataset.

H5D_WRITE, *Dataset_id*, *Data* [, MEMORY_SPACE_ID=*value*] [, FILE_SPACE_ID=*value*]

H5F_CLOSE - Closes the specified file and releases resources used by it.

H5F_CLOSE, *File_id*

H5F_CREATE - The primary function for creating HDF5 files.

Result = H5F_CREATE(*Filename*)

H5F_IS_HDF5 - Determines if a file is in the HDF5 format.

Result = H5F_IS_HDF5(*Filename*)

H5F_OPEN - Opens an existing HDF5 file.

Result = H5F_OPEN(*Filename*)([, /WRITE])

H5G_CLOSE - Closes the specified group and releases resources used by it.

H5G_CLOSE, *Group_id*

H5G_CREATE - Creates a new empty group and gives it a name..

Result = H5G_CREATE(*Loc_id*, *Name*)

H5G_GET_COMMENT - Retrieves a comment string from a specified object.

Result = H5G_GET_COMMENT(*Loc_id*, *Name*)

H5G_GET_LINKVAL - Returns the name of the object pointed to by a symbolic link.

Result = H5G_GET_LINKVAL(*Loc_id*, *Name*)

H5G_GET_MEMBER_NAME - Retrieves the name of an object within a group, by its zero-based index.

Result = H5G_GET_MEMBER_NAME(*Loc_id*, *Name*, *Index*)

H5G_GET_NMEMBERS - Returns the number of objects within a group.

Result = H5G_GET_NMEMBERS(*Loc_id*, *Name*)

H5G_GET_NUM_OBJS - Returns number of objects in the group specified by its identifier.

Result = H5G_GET_NUM_OBJS(*Loc_id*)

H5G_GET_OBJ_NAME_BY_IDX - Returns a name of an object specified by an index.

Result = H5G_GET_OBJ_NAME_BY_IDX(*Loc_id*, *Index*)

H5G_GET_OBJINFO - Retrieves information from a specified object.

Result = H5G_GET_OBJINFO(*Loc_id*, *Name* [, /FOLLOW_LINK])

H5G_LINK - Creates a link of the specified type. A link can only point to one of the three classes of named objects: group, dataset, and named datatype.

H5G_LINK, *Loc_id*, *Current_Name*, *New_Name* [, /SOFTLINK] [, NEW_LOC_ID=*value*]

H5G_MOVE - Renames/moves an object within an HDF5 group or file.

H5G_MOVE, *Loc_id*, *Src_Name*, *Dst_Name* [, NEW_LOC_ID=*value*]

H5G_OPEN - Opens an existing group within an HDF5 file.

Result = H5G_OPEN(*Loc_id*, *Name*)

H5G_SET_COMMENT - Sets a comment for a specified object.

H5G_SET_COMMENT, *Loc_id*, *Name*, *Comment*

H5G_UNLINK - Removes the link to an object from a group.

H5G_UNLINK, *Loc_id*, *Name*

H5I_GET_FILE_ID - Retrieves an identifier for the file containing the specified object.

Result = H5I_GET_FILE_ID(*Loc_id*)

H5I_GET_TYPE - Returns the object's type.

Result = H5I_GET_TYPE(*Obj_id*)

H5R_CREATE - Creates a reference to either an object or a dataspace region of a dataset.

Result = H5R_CREATE(*Loc_id*, *Name* [,DATASPACE_ID=*value*])

H5R_DEREFERENCE - Opens a reference and returns the object identifier.

Result = H5R_DEREFERENCE(*Loc_id*, *Reference*)

H5R_GET_OBJECT_TYPE - Returns the type of object that an object reference points to.

Result = H5R_GET_OBJECT_TYPE(*Loc_id*, *Reference*)

H5R_GET_REGION - Retrieves a dataspace associated with a region reference.

Result = H5R_GET_REGION(*Dataset_id*, *Reference*)

H5S_CLOSE - Releases and terminates access to a dataspace.

H5S_CLOSE, *Dataspace_id*

H5S_COPY - Copies an existing dataspace.
Result = H5S_COPY(*Dataspace_id*)

H5S_CREATE_SCALAR - Creates a scalar dataspace.
Result = H5S_CREATE_SCALAR()

H5S_CREATE_SIMPLE - Creates a simple dataspace.
Result = H5S_CREATE_SIMPLE(*Dimensions* [, MAX_DIMENSIONS=*vector*])

H5S_GET_SELECT_BOUNDS - Retrieves the coordinates of the bounding box containing the current dataspace selection.
Result = H5S_GET_SELECT_BOUNDS(*Dataspace_id*)

H5S_GET_SELECT_ELEM_NPOINTS - Determines the number of element points in the current dataspace selection.
Result = H5S_GET_SELECT_ELEM_NPOINTS (*Dataspace_id*)

H5S_GET_SELECT_ELEM_POINTLIST - Returns a list of the element points in the current dataspace selection.
Result = H5S_GET_SELECT_ELEM_POINTLIST (*Dataspace_id* [, START=*value*] [, NUMBER=*value*])

H5S_GET_SELECT_HYPER_BLOCKLIST - Returns a list of the hyperslab blocks in the current dataspace selection.
Result = H5S_GET_SELECT_HYPER_BLOCKLIST (*Dataspace_id* [, START=*value*] [, NUMBER=*value*])

H5S_GET_SELECT_HYPER_NBLOCKS - Determines the number of hyperslab blocks in the current dataspace selection.
Result = H5S_GET_SELECT_HYPER_NBLOCKS (*Dataspace_id*)

H5S_GET_SELECT_NPOINTS - Determines the number of elements in a dataspace selection.
Result = H5S_GET_SELECT_NPOINTS (*Dataspace_id*)

H5S_GET_SIMPLE_EXTENT_DIMS - Returns the dimension sizes for a dataspace.
Result = H5S_GET_SIMPLE_EXTENT_DIMS(*Dataspace_id* [, MAX_DIMENSIONS=*variable*])

H5S_GET_SIMPLE_EXTENT_NDIMS - Determines the number of dimensions (or rank) of a dataspace.
Result = H5S_GET_SIMPLE_EXTENT_NDIMS (*Dataspace_id*)

H5S_GET_SIMPLE_EXTENT_NPOINTS - Determines the number of elements in a dataspace.
Result = H5S_GET_SIMPLE_EXTENT_NPOINTS (*Dataspace_id*)

H5S_GET_SIMPLE_EXTENT_TYPE - Returns the current class of a dataspace.
Result = H5S_GET_SIMPLE_EXTENT_TYPE (*Dataspace_id*)

H5S_IS_SIMPLE - Determines whether a dataspace is a simple dataspace.
Result = H5S_IS_SIMPLE(*Dataspace_id*)

H5S_OFFSET_SIMPLE - Sets the selection offset for a simple dataspace.
H5S_OFFSET_SIMPLE, *Dataspace_id*, *Offset*

H5S_SELECT_ALL - Selects the entire extent of a dataspace.
H5S_SELECT_ALL, *Dataspace_id*

H5S_SELECT_ELEMENTS - Selects array elements to be included in the selection for a dataspace.
H5S_SELECT_ELEMENTS, *Dataspace_id*, *Coordinates*, /RESET

H5S_SELECT_HYPERSLAB - Selects a hyperslab region to be included in the selection for a dataspace.
H5S_SELECT_HYPERSLAB, *Dataspace_id*, *Start*, *Count* [, BLOCK=*vector*] [, /RESET] [, STRIDE=*vector*]

H5S_SELECT_NONE - Resets the dataspace selection region to include no elements.
H5S_SELECT_NONE, *Dataspace_id*

H5S_SELECT_VALID - Verifies that the selection is within the extent of a dataspace.
Result = H5S_SELECT_VALID(*Dataspace_id*)

H5S_SET_EXTENT_NONE - Removes the extent of a dataspace and sets the type to H5S_NO_CLASS. As such the dataspace cannot be resized or used in the creation of datasets or attributes.
H5S_SET_EXTENT_NONE, *Dataspace_id*

H5S_SET_EXTENT_SIMPLE - Sets or resets the extent of a dataspace.
H5S_SET_EXTENT_SIMPLE, *Dataspace_id*, *Dimensions* [,MAX_DIMENSIONS=*vector*]

H5T_ARRAY_CREATE - Creates and array datatype object.
Result = H5T_ARRAY_CREATE(*Datatype_id*, *Dimensions*)

H5T_CLOSE - Releases the specified datatype's identifier and releases resources used by it.
H5T_CLOSE, *Datatype_id*

H5T_COMMIT - Commits a transient datatype to a file, creating a new named datatype.
H5T_COMMIT, *Loc_id*, *Name*, *Datatype_id*

H5T_COMMITTED - Determines whether a datatype is a named datatype or a transient type.
Result = H5T_COMMITTED(*Datatype_id*)

H5T_COPY - Copies an existing datatype.
Result = H5T_COPY(*Datatype_id*)

H5T_EQUAL - Determines whether two datatype identifiers refer to the same datatype.
Result = H5T_EQUAL(*Datatype_id1*, *Datatype_id2*)

H5T_GET_ARRAY_DIMS - Returns the dimension sizes for an array datatype object.
Result = H5T_GET_ARRAY_DIMS(*Datatype_id* [, PERMUTATIONS=*variable*])

H5T_GET_ARRAY_NDIMS - Determines the number of dimensions (or rank) of an array datatype object.
Result = H5T_GET_ARRAY_NDIMS(*Datatype_id*)

H5T_GET_CLASS - Returns the datatype's class.
Result = H5T_GET_CLASS(*Datatype_id*)

H5T_GET_CSET - Returns the character set type of a string datatype.
Result = H5T_GET_CSET(*Datatype_id*)

H5T_GET_EBIAS - Returns the exponent bias of a floating-point type.
Result = H5T_GET_EBIAS(*Datatype_id*)

H5T_GET_FIELDS - Retrieves information about the positions and sizes of bit fields within a floating-point datatype.
Result = H5T_GET_FIELDS(*Datatype_id*)

H5T_GET_INPAD - Returns the padding method for unused internal bits within a floating-point datatype.
Result = H5T_GET_INPAD(*Datatype_id*)

H5T_GET_MEMBER_CLASS - Returns the datatype class of a compound datatype member.
Result = H5T_GET_MEMBER_CLASS(*Datatype_id*, *Member*)

H5T_GET_MEMBER_NAME - Returns the datatype name of a compound datatype member.
Result = H5T_GET_MEMBER_NAME(*Datatype_id*, *Member*)

H5T_GET_MEMBER_OFFSET - Returns the byte offset of a field within a compound datatype.
Result = H5T_GET_MEMBER_OFFSET(*Datatype_id*, *Member*)

H5T_GET_MEMBER_TYPE - Returns the datatype identifier for a specified member within a compound datatype.
Result = H5T_GET_MEMBER_TYPE(*Datatype_id*, *Member*)

H5T_GET_NMEMBERS - Returns the number of fields in a compound datatype.
Result = H5T_GET_NMEMBERS(*Datatype_id*)

H5T_GET_NORM - Returns the mantissa normalization of a floating-point datatype.
Result = H5T_GET_NORM(*Datatype_id*)

H5T_GET_OFFSET - Returns the bit offset of the first significant bit in an atomic datatype.
Result = H5T_GET_OFFSET(*Datatype_id*)

H5T_GET_ORDER - Returns the byte order of an atomic datatype.
Result = H5T_GET_ORDER(*Datatype_id*)

H5T_GET_PAD - Returns the padding method of the least significant bit (*lsb*) and most significant bit (*msb*) of an atomic datatype.
Result = H5T_GET_PAD(*Datatype_id*)

H5T_GET_PRECISION - Returns the precision in bits of an atomic datatype.
Result = H5T_GET_PRECISION(*Datatype_id*)

H5T_GET_SIGN - Returns the sign type for an integer datatype.
Result = H5T_GET_SIGN(*Datatype_id*)

H5T_GET_SIZE - Returns the size of a datatype in bytes.
Result = H5T_GET_SIZE(*Datatype_id*)

H5T_GET_STRPAD - Returns the padding method for a string datatype.
Result = H5T_GET_STRPAD(*Datatype_id*)

H5T_GET_SUPER - Returns the base datatype from which a datatype is derived.
Result = H5T_GET_SUPER(*Datatype_id*)

H5T_IDLTYPE - Returns the IDL type code corresponding to a datatype.
Result = H5T_IDLTYPE(*Datatype_id*
[, ARRAY_DIMENSIONS=*variable*]
[, STRUCTURE=*variable*])

H5T_IDL_CREATE - Creates a datatype object based on the IDL type of the supplied data.
Result = H5T_IDL_CREATE(*Data*
[,MEMBER_NAMES=*vector*])

H5T_INSERT - Adds a new member to the end of a compound datatype.
H5T_INSERT, *Datatype_id*, *Name*, *Field_id*

H5T_MEMTYPE - Returns the native memory datatype corresponding to a file datatype.
Result = H5T_MEMTYPE(*Datatype_id*)

H5T_OPEN - Opens a named datatype.
Result = H5T_OPEN(*Loc_id*, *Name*)

H5T_REFERENCE_CREATE - Creates a reference datatype object.
Result = H5T_REFERENCE_CREATE([/REGION])

NetCDF Routines

NCDF_ATTCOPY - Copies attribute from one netCDF file to another.
Result = NCDF_ATTCOPY(*Incdf* [, *Invar* | , /IN_GLOBAL] , *Name*, *Outcdf* [, *Outvar*] [, /OUT_GLOBAL])

NCDF_ATTDEL - Deletes an attribute from a netCDF file.
NCDF_ATTDEL, *Cdfid* [, *Varid* | , /GLOBAL] , *Name*

NCDF_ATTGET - Retrieves value of an attribute from a netCDF file.
NCDF_ATTGET, *Cdfid* [, *Varid* | , /GLOBAL] , *Name*, *Value*

NCDF_ATTINQ - Returns information about a netCDF attribute.
Result = NCDF_ATTINQ(*Cdfid* [, *Varid* | , /GLOBAL] , *Name*)

NCDF_ATTNAME - Returns the name of an attribute given its ID.
Result = NCDF_ATTNAME(*Cdfid* [, *Varid* | , /GLOBAL] , *Attnum*)

NCDF_ATTPUT - Creates an attribute in a netCDF file.

NCDF_ATTPUT, *Cdfid* [, *Varid* | , /GLOBAL] , *Name* , *Value* [, LENGTH=*value*] [, /BYTE | , /CHAR | , /DOUBLE | , /FLOAT | , /LONG | , /SHORT]

NCDF_ATTRENAME - Renames an attribute in a netCDF file.

NCDF_ATTRENAME, *Cdfid* [, *Varid* | , /GLOBAL] *Oldname*, *Newname*

NCDF_CLOSE - Closes an open netCDF file.

NCDF_CLOSE, *Cdfid*

NCDF_CONTROL - Performs miscellaneous netCDF operations.

NCDF_CONTROL, *Cdfid* [, /ABORT] [, /ENDEF] [, /FILL | , /NOFILL] [, /NOVERBOSE | , /VERBOSE] [, OLDFILL=*variable*] [, /REDEF] [, /SYNC]

NCDF_CREATE - Creates a new netCDF file.

Result = NCDF_CREATE(*Filename* [, /CLOBBER | , /NOCLOBBER])

NCDF_DIMDEF - Defines a dimension given its name and size.

Result = NCDF_DIMDEF(*Cdfid*, *DimName*, *Size* [, /UNLIMITED])

NCDF_DIMID - Returns the ID of a netCDF dimension, given the name of the dimension.

Result = NCDF_DIMID(*Cdfid*, *DimName*)

NCDF_DIMINQ - Retrieves the name and size of a dimension in a netCDF file, given its ID.

NCDF_DIMINQ, *Cdfid*, *Dimid*, *Name*, *Size*

NCDF_DIMRENAME - Renames an existing dimension in a netCDF file that has been opened for writing.

NCDF_DIMRENAME, *Cdfid*, *Dimid*, *NewName*

NCDF_EXISTS - Returns True if the netCDF format library is supported on the current IDL platform.

Result = NCDF_EXISTS( )

NCDF_INQUIRE - Returns information about an open netCDF file.

Result = NCDF_INQUIRE(*Cdfid*)

NCDF_OPEN - Opens an existing netCDF file.

Result = NCDF_OPEN(*Filename* [, /NOWRITE | , /WRITE])

NCDF_VARDEF - Adds a new variable to an open netCDF file in define mode.

Result = NCDF_VARDEF(*Cdfid*, *Name* [, *Dim*] [, /BYTE | , /CHAR | , /DOUBLE | , /FLOAT | , /LONG | , /SHORT])

NCDF_VARGET - Retrieves a hyperslab of values from a netCDF variable.

NCDF_VARGET, *Cdfid*, *Varid*, *Value* [, COUNT=*vector*] [, OFFSET=*vector*] [, STRIDE=*vector*]

NCDF_VARGET1 - Retrieves one element from a netCDF variable.

NCDF_VARGET1, *Cdfid*, *Varid*, *Value* [, OFFSET=*vector*]

NCDF_VARID - Returns the ID of a netCDF variable.

Result = NCDF_VARID(*Cdfid*, *Name*)

NCDF_VARINQ - Returns information about a netCDF variable, given its ID.

Result = NCDF_VARINQ(*Cdfid*, *Varid*)

NCDF_VARPUT - Writes a hyperslab of values to a netCDF variable.

NCDF_VARPUT, *Cdfid*, *Varid*, *Value* [, COUNT=*vector*] [, OFFSET=*vector*] [, STRIDE=*vector*]

NCDF_VARRENAME - Renames a netCDF variable.

NCDF_VARRENAME, *Cdfid*, *Varid*, *Name*